THE POWER TO KNOW®

SAS/Warehouse Administrator® 2.0 User's Guide

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2002. *SAS/Warehouse Administrator® 2.0 User's Guide*. Cary, NC: SAS Institute Inc.

SAS/Warehouse Administrator® 2.0 User's Guide

Copyright © 2002, SAS Institute Inc., Cary, NC, USA

ISBN 978-1-59047-579-9

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

1st electronic book, February 2009

1st printing, June 2002
2nd printing, March 2009

Contents

PART 1

Introduction

CHAPTER

1

Using This Document

Purpose of This Document

This document, together with the online help, describes how to build data warehouses with SAS/Warehouse Administrator software. The document describes data warehousing concepts and gives examples of how warehouse elements can be implemented. The online help describes SAS/Warehouse Administrator windows, and it explains the general steps for creating groups, data stores, and processes.

Intended Audience

This document is intended for data warehouse administrators who have a thorough knowledge of Base SAS software on UNIX or PC platforms. Here are the main SAS features and products that you will need to know:

- SAS engines
- LIBNAME statement
- FILE and INFILE statements
- DATA step views
- PROC SQL views
- SAS/ACCESS software (PROC ACCESS and PROC DBLOAD)
- SAS/CONNECT software.

For details about these features and products, see the online documentation for Base SAS, SAS/ACCESS, and SAS/CONNECT, as well as the relevant hardcopy documents in the SAS publications catalog.

Other SAS/Warehouse Administrator Documentation

The online help for SAS/Warehouse Administrator describes its windows and summarizes the main tasks that you can perform using the product. There are several ways to display the online help.

To display the table of contents for SAS/Warehouse Administrator help:

1 Run SAS.
2 From the menu bar, select **`Help`**, then **`SAS System Help`**.
3 In the left panel of the help window, open **`Help on SAS Software Products`**, then scroll down to the SAS/Warehouse Administrator topic and expand it.

To display the help for an active window, click its Help button. If the window does not have a Help button, from the SAS menu bar, select **`Help`**, then **`Using This Window`**.

To display task-oriented topics that are associated with certain SAS/Warehouse Administrator windows:

1 Display help for the window as described above.
2 Scroll down to the bottom of the topic. Some windows have a link to a "Maintaining..." topic, which summarizes how to perform tasks that are associated with the window.

An online tutorial entitled *Getting Started with SAS/Warehouse Administrator Software* is available from the SAS Web site. This tutorial shows you how to add a new Warehouse Environment and walks you through creating a sample Data Warehouse.

Here are two ways to access the tutorial:

□ Run SAS/Warehouse Administrator. From the desktop, position the cursor on the Getting Started icon, click the right mouse button and select **`Run`**.
□ Run SAS/Warehouse Administrator. From the desktop, open a Warehouse Environment in the SAS/Warehouse Administrator Explorer. In the Explorer, from the menu bar, select **`Help`**, then **`Getting Started with SAS/Warehouse Administrator Software`**.

If you want to write programs that read or write SAS/Warehouse Administrator metadata without using the user interface, see *SAS/Warehouse Administrator 2.3 Metadata API Reference*.

CHAPTER 2 Overview of SAS/Warehouse Administrator

What Is a Data Warehouse?

A *data warehouse* is a collection of data that is extracted from one or more sources for the purpose of query and analysis. For example, a toy company might create a data warehouse that integrates sales, product, and customer information from various sources. This collection would help the company analyze how toy sales are affected by geography, by different promotions, by the gender and age of the customer, and by other factors.

A *data mart* is a limited data warehouse that is often designed to meet the needs of a particular department or individual. A data mart is more limited in scope than a data warehouse, which typically contains information used by more than one department.

Note: Terms that are in *italics* are defined in the glossary. △

What Is SAS/Warehouse Administrator?

SAS/Warehouse Administrator is an application that provides a visual environment for managing data warehouses. Using the windows in this application, you can specify *metadata* that defines data sources, data stores, code libraries, and other warehouse resources. SAS/Warehouse Administrator then uses this metadata to generate or retrieve the code that extracts, transforms, and loads the data into your warehouse.

Through its metadata, SAS/Warehouse Administrator provides a single point of control for managing

- *data sources*, on any platform accessible to SAS, in any format accessible to SAS
- *data stores*, on any platform accessible to SAS, in any format accessible to SAS
- *Process Flows*, which define how data moves from sources to targets
- source code generated by SAS/Warehouse Administrator
- user-written source code retrieved from code libraries
- *Jobs*, which execute the code that moves data from sources to targets.

SAS/Warehouse Administrator also provides

- a *Process Library*, which contains routines that can be included in the Process Flows for data stores in your warehouse. For example, there are routines that standardize addresses, and there are routines that generate the code required to load a warehouse table into a DBMS, such as Oracle, using native loading software.
- Metadata Exporters, which export SAS/Warehouse Administrator metadata to other applications.
- a *Metadata API*, which enables you to write client applications that read or write SAS/Warehouse Administrator metadata.

You can also download SAS/Warehouse Administrator tools and utilities from the SAS Web site. Here are a few examples:

- Add-in tools that generate the metadata for a table by reading a data model, or that trace the impact of a change in a warehouse data store.
- The MetaSpace Explorer, which is a Java applet that enables you to browse metadata that has been exported by SAS/Warehouse Administrator.
- Publish and Subscribe add-ins that enable SAS/Warehouse Administrator to publish a package of information whenever a *Subject* or similar group is updated. You can now publish a package of information
  - directly to email account(s)
  - directly to message queue(s)
  - to subscribers of one or more channels (associated with a warehouse object or explicitly specified with the defined package add-in)
  - to an archive directory (useful with for historical snapshots).

For details about many of the features previously described, see the online help. To view the online help for these features, from the left panel of the SAS Help window, select **`Help on SAS Software Products`**, then **`SAS/Warehouse Administrator`**, then **`Using SAS/Warehouse Administrator`**, then **`Overview`**, then **`Overview of SAS/Warehouse Administrator`**.

Managing through Metadata

Metadata is a definition or description of data. The physical metadata for a table might specify a certain number of rows and columns, with certain transformations applied to some of the columns. The business metadata for a 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.

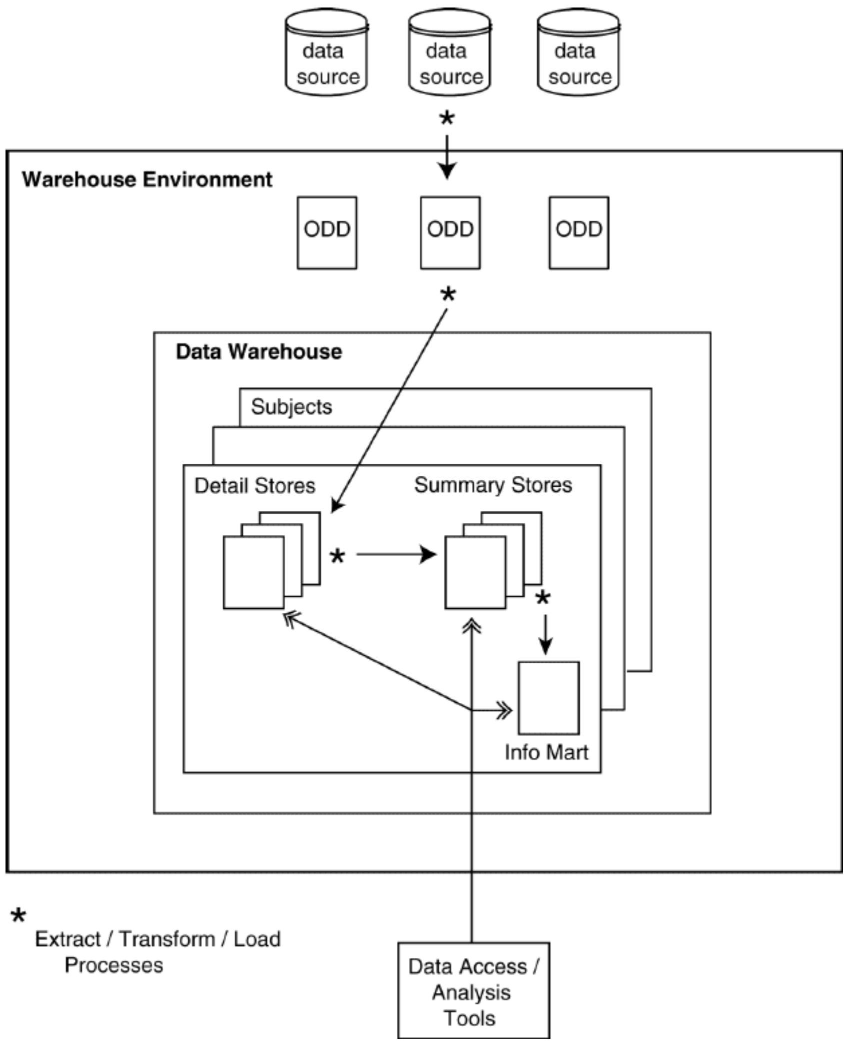
SAS/Warehouse Administrator uses the metadata that you enter to generate or retrieve the code that creates your data warehouse. This metadata-driven approach might seem inefficient when compared to writing a program to perform a given task. However, SAS/Warehouse Administrator offers the following advantages over writing and managing individual programs:

- a single point of control for managing data warehouse resources because the resources are defined in SAS/Warehouse Administrator metadata.
- a consistent and documented flow of information across computing platforms.
- because the data flows are documented, you can trace information from its source through the entire data warehouse. Documentation also makes it easier to analyze and improve data flows, resulting in better data quality.
- metadata can be used to automate many tasks. For example, if you have entered the appropriate metadata, you can have SAS/Warehouse Administrator generate the code to create and load a data store. If you have entered the appropriate metadata, you can use the Impact Analysis add-in to trace the impact of a proposed change to a data store.
- metadata can be used as a “view” on the data warehouse. For example, you can export the metadata for a data warehouse and make it available to business analysts, who might need to answer questions, such as What is the origin of the data in this report? What does this column mean? How is it derived?.

SAS/Warehouse Administrator Data Flow

The following figure illustrates the flow of information through a data warehouse managed by SAS/Warehouse Administrator.

Figure 2.1 Data Flow Through a Warehouse



The methodology for the data flow illustrated in Figure 2.1 might be summarized as follows:

1 Information is extracted from data sources and is stored in a table or view that is registered in a *Warehouse Environment*. The table or view is called an *Operational Data Definition* (ODD).

2 The ODDs are used as inputs to data stores. The data stores are organized under grouping elements called *Data Warehouses* and *Subjects*.

3 The detail data stores can be used as inputs to summary data stores.

4 Within SAS/Warehouse Administrator, you can use *Information Marts* to generate output from the detail data stores and the summary data in the warehouse. For example, an *InfoMart Item* might be used to display a chart that is generated from summary data.

5 After the data stores are available in the data warehouse, you can exploit them using SAS software or other software products.

For example, you could use SAS Enterprise Miner to analyze patterns in detail data. You could use SAS/EIS software to analyze multidimensional summary data, and you could have a link from a Web page to an InfoMart Item that displays a chart that is generated from summary data.

Tour of the Main Windows

The following table lists the main windows in SAS/Warehouse Administrator. Each window is briefly described in the sections that follow.

Table 2.1 Main SAS/Warehouse Administrator Windows

Window	Description
"Desktop" on page 9	Initial window for SAS/Warehouse Administrator. Used to add or open Warehouse Environments.
"Explorer" on page 10	Used to add, edit, and browse the properties of groups and data stores.
"Define Items Used Globally Window" on page 11	Used to manage host definitions and other metadata shared within a Warehouse Environment.
"Process Editor" on page 12	Used to define Jobs and Process Flows which define how data moves from sources to targets.
"Job Properties" on page 13	Used to define Jobs which execute the code that moves data from sources to targets.
"Load Generation/Execution Properties" on page 14	Used to generate and manage SAS code for a Job.

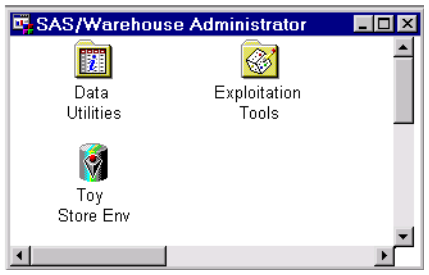
Displaying Help for the Main Windows

For full details about each of the windows that are listed in Table 2.1 on page 9, see the online help. To display the relevant online help, in the SAS Help contents for **`Using SAS/Warehouse Administrator Software`**, select **`Overview`**, then **`Main SAS/Warehouse Administrator Windows`**.

Desktop

To display the desktop for SAS/Warehouse Administrator, run SAS on a machine where SAS/Warehouse Administrator has been installed, then type **`dw`** on the command line. The desktop will look similar to the one shown in Display 2.1 on page 10.

Display 2.1 Desktop for SAS/Warehouse Administrator



The default desktop includes folders for Data Utilities and Exploitation Tools, which are described under “Starting SAS/Warehouse Administrator” on page 18. If any *Data Warehouse Environments* have been defined for your site, their icons will appear on the desktop. Think of a Warehouse Environment as a container for some of the metadata associated with one or more data warehouses. The Toy Store Env icon in Figure 2.1 represents the main example Environment described in this document.

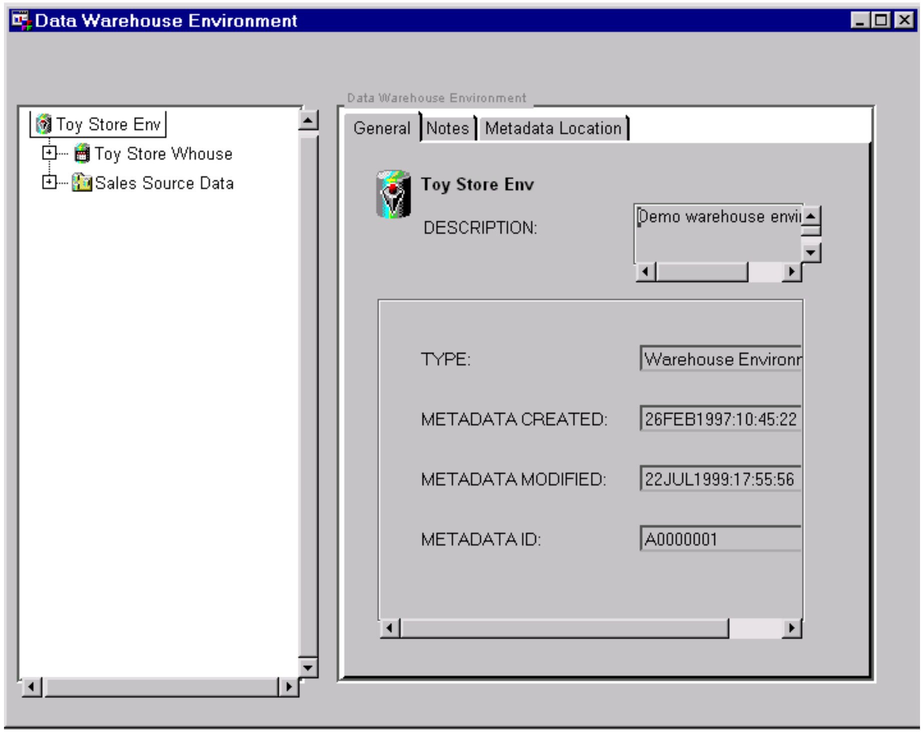
For details about the SAS/Warehouse Administrator desktop, display its online help as described in “Displaying Help for the Main Windows” on page 9. See also “Starting SAS/Warehouse Administrator” on page 18.

Explorer

To display the Explorer window, display the SAS/Warehouse Administrator desktop, position the cursor on an Environment icon, click the right mouse button, and select **`Edit`** from the pop-up menu. (Under Microsoft Windows and OS/2 operating environments, you can simply double-click the Environment icon.)

When you first open an Environment in the Explorer, it might look similar to Display 2.2 on page 11.

Display 2.2 Explorer with Toy Store Environment, Unexpanded



DESCRIPTION: Demo warehouse envi

TYPE: Warehouse Environ

METADATA CREATED: 26FEB1997:10:45:22

METADATA MODIFIED: 22JUL1999:17:55:56

METADATA ID: A0000001

The Explorer is used to define and browse the metadata for groups and data stores in a Warehouse Environment. SAS/Warehouse Administrator uses this metadata to generate or retrieve the code that extracts, transforms, and loads the data into your data stores.

The white area at left displays the hierarchy of groups and data stores in the Explorer. The tabs on the right display the metadata for the element that has been selected with the left mouse button. For example, in Display 2.2 on page 11, the metadata for the Toy Store Warehouse Environment is displayed.

In the Explorer hierarchy, the main groups are indented under the Environment, and other groups or data stores are indented under the main groups. For example, in Display 2.2 on page 11, there are two main groups indented under the Toy Store Warehouse Environment: a Data Warehouse (Toy Store Whouse), and an ODD Group (Sales Source Data).

For details about the SAS/Warehouse Administrator Explorer, refer to its online help.

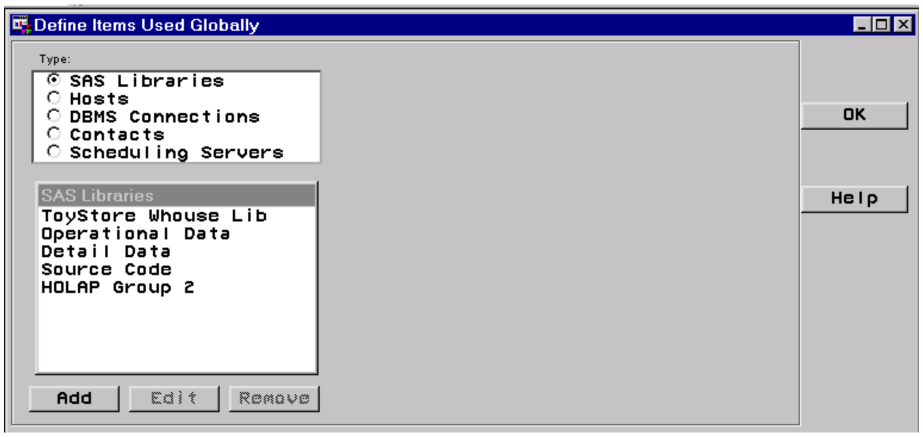
Define Items Used Globally Window

To display the Define Items Used Globally window, open an Environment in the Explorer, then select

File ► Setup

from the pull-down menu above the Explorer. A window will be displayed that looks similar to the one shown in Display 2.3 on page 12.

Display 2.3 Define Items Used Globally Window



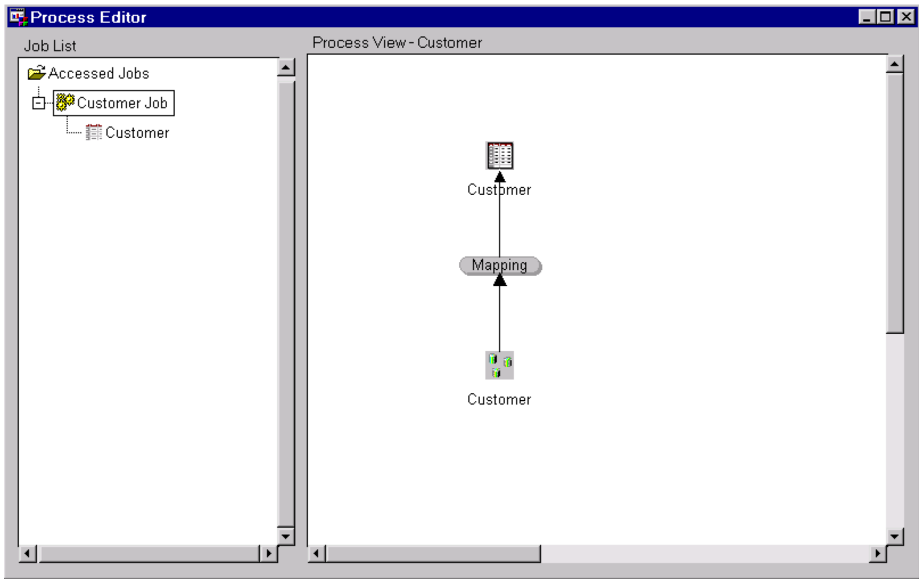
The Define Items Used Globally window is used to define *global metadata* for hosts, libraries, and other resources that are shared at the Environment level. By adding metadata records for these resources, you register them in the current Environment. After these records have been saved, you can include them in the metadata for groups, data stores, processes, Jobs, or other objects in the current Environment.

For details about the Define Items Used Globally window, refer to its online help.

Process Editor

One way to display the Process Editor is to open an Environment in the Explorer, click a data store with the right mouse button, and select **`Process`** from the pop-up menu. The Process Editor is used to manage *Jobs*, which specify the processes and Process Flows that create one or more data stores. Display 2.4 on page 12 illustrates the Job and Process Flow for the Customer table.

Display 2.4 Process Editor



In Display 2.4 on page 12, the Job for Customer is represented by the icon with the rectangle around it in the left panel. A Job is a metadata record that specifies the processes that create one or more data stores. It enables you to connect a series of process steps into a single unit. A Job may include scheduling metadata that enables the processes to be executed in batch mode at a specified date and time.

In Display 2.4 on page 12, the *Process Flow* for Customer appears in the right panel. A Process Flow is a diagram that is composed of symbols, with connecting arrows and descriptive text, that illustrate the sequence of each process associated with the Job that is selected in the left panel 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.

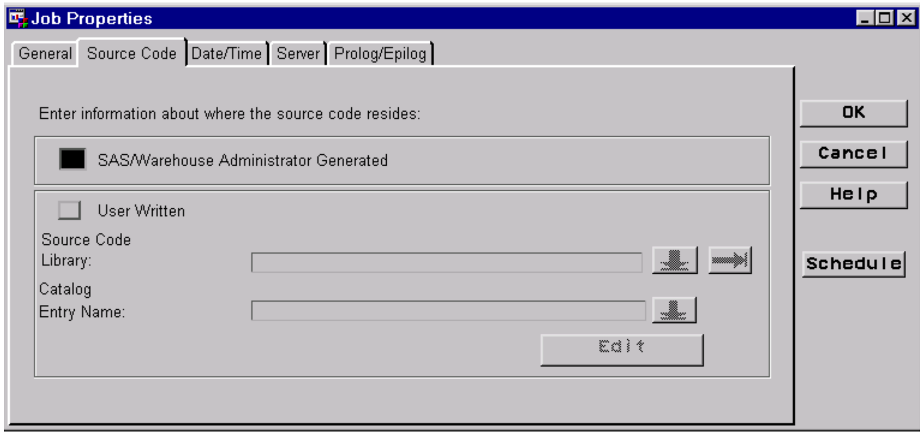
Keep in mind that in a Process Flow, data moves from the bottom to the top. The top icon is the output table that is created by the active Job (Job selected in the left panel of the Process Editor). For example, in Display 2.4 on page 12, data moves from the source data at the bottom of the flow, through a Mapping process, to the Customer table at the top. Between the source and the target, columns can be added, data can be scrubbed, and other transformations can take place.

For details about the Process Editor, refer to its online help.

Job Properties

To display the properties window for an existing Job, display the Job in the Process Editor as shown in Display 2.4 on page 12. Click the Job with the right mouse button and select **`Properties`** from the pop-up menu. The Job Properties window displays. Display 2.5 on page 13 shows the Source Code tab of this window.

Display 2.5 Job Properties Window



Enter information about where the source code resides:

■ SAS/Warehouse Administrator Generated

☐ User Written

Source Code Library:

Catalog Entry Name:

In Display 2.5 on page 13, note that the **`SAS/Warehouse Administrator Generated`** option is selected. If this option is selected, SAS/Warehouse Administrator will use the Process Flow associated with the active Job to generate source code for the Job. For example, if the **`SAS/Warehouse Administrator Generated`** option was selected for the Customer Job shown in Display 2.4 on page 12, SAS/Warehouse Administrator would use the Process Flow shown in that figure to generate source code for the Job.

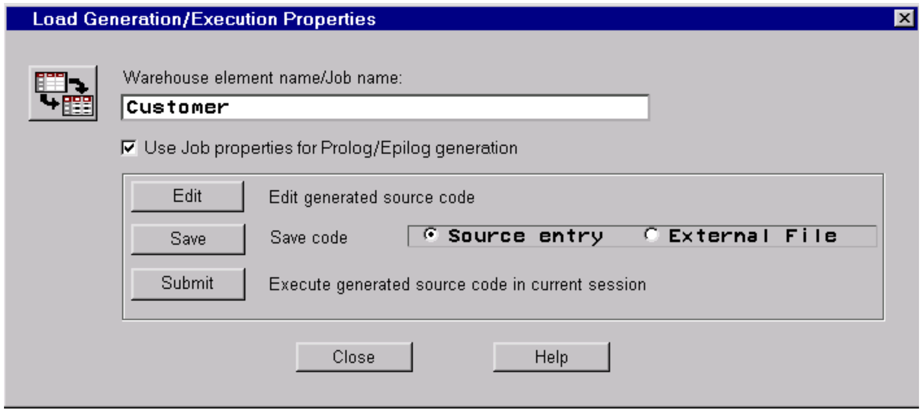
If the **`SAS/Warehouse Administrator Generated`** option is not selected, you must specify the location of user-written source code for the Job.

The Date/Time tab, Server tab, and Prolog/Epilog tab on this window can be used to enter scheduling metadata for the Job. For details about the Job Properties window, refer to its online help.

Load Generation/Execution Properties

To display the Load Generation/Execution Properties window, open a data store in the Process Editor. In the left panel, click the Job for the data store with the right mouse button and select **Run** from the pop-up menu. The Load Generation/Execution Properties window displays. It will look similar to the one shown in Display 2.6 on page 14.

Display 2.6 Load Generation/Execution Properties Window


Load Generation/Execution Properties

Warehouse element name/Job name: Customer

[✓] Use Job properties for Prolog/Epilog generation

Edit — Edit generated source code

Save — Save code [✓ Source entry] External File

Submit — Execute generated source code in current session

Close Help

From this window, you can submit, save, or edit the code generated for the Job you selected in the Process Editor. For details about the Load Generation/Execution Properties window, refer to its online help.

Task Summaries

Many different approaches can be taken when building a data warehouse. Here are some general guidelines:

- plan from the top down — identify your business goals, then identify the outputs, sources, and data stores that are required to support these goals
- implement from the bottom up — from source data, to detail data stores in a subject area, to summary data stores in a subject area
- build your warehouse one subject area at a time
- define, load, and test warehouse data sources and data stores as you go.

Plan Your Data Warehouse

1. Identify the business problem(s) to be solved.
2. Identify the subject areas to be included in your data warehouse (such as Sales, Products, Customers, and so on).

3 Choose a subject area to be developed. The first subject area should be important enough to illustrate the value of your project, but simple enough so that it can be developed quickly and successfully.

4 Draft the reports and other outputs that you expect to get from the chosen subject area.

5 Identify data sources required to produce the reports and other outputs that you expect to get from the chosen subject area.

6 Identify the main column mappings and data transformations between sources and targets in the chosen subject area.

7 Select the hardware and software required to access the data sources, execute the transformations, and store the refined data in the appropriate formats and locations.

8 Choose the SAS/Warehouse Administrator data stores that are appropriate for the chosen subject area.

9 Choose the appropriate exploitation tool(s) for the chosen subject area (such as SAS/EIS for multidimensional analysis; Enterprise Miner for data mining, and so on).

A detailed explanation of how to create a project plan for your data warehouse is beyond the scope of this document. However, the following chapters will help you with the SAS/Warehouse Administrator portion of such a plan:

□ Chapter 3, "Planning Your Hardware and Software," on page 21

□ Chapter 4, "Planning Your Data Stores and Processes," on page 41

Create a Data Warehouse Environment

1 Create a directory structure for the Data Warehouse Environment.

2 If necessary, update the SAS configuration file and the SAS autoexec file that is used during the SAS/Warehouse Administrator session so that any required options or librefs are available.

3 Add a Data Warehouse Environment.

   a Define global metadata for the Environment.

   b Register data sources for the Environment (create Operational Data Definitions).

      i Enter metadata for the columns and location of the data sources.

      ii Define one or more Jobs for the data sources.

      iii Execute the Job(s) for the data sources.

      iv Verify that the data sources are available.

For details about the tasks in this section, see the following chapters:

□ Chapter 5, "Maintaining Environments," on page 61

□ Chapter 6, "Maintaining Global Metadata," on page 75

□ Chapter 7, "Registering Data Sources," on page 107

If you need to convert a Data Warehouse Environment created with an earlier version of SAS/Warehouse Administrator, see Appendix 1, "Converting Metadata for Environments and Warehouses," on page 323.

Create a Data Warehouse

1 Add a Data Warehouse within the Environment.

2 Create a Subject within the Warehouse. Name it after a subject area identified in your project plan.

3 Create appropriate detail data stores within the Subject.

  a Enter metadata for the data sources, columns, and locations of the detail data stores.
  b Define one or more Jobs for the detail data stores.
  c Execute the Job(s) for the detail data stores.
  d Verify that the detail data stores are available.

4 If required by your project plan, create appropriate summary data stores within the Subject.

  a Enter metadata for the data sources, columns, and locations of the summary data stores.
  b Define one or more Jobs for the summary data stores.
  c Execute the Job(s) for the summary data stores.
  d Verify that the summary data stores are available.

5 If required by your project plan, create appropriate Information Marts in the Subject.

  a Enter metadata for the InfoMart Items and InfoMart Files.
  b Define one or more Jobs for these objects.
  c Execute the Job(s).
  d Verify that the InfoMart Items and InfoMart Files are available.

For details about the tasks in this section, see the following chapters:

- Chapter 8, “Maintaining Data Warehouses and Subjects,” on page 133
- Chapter 9, “Maintaining Data Tables,” on page 145
- Chapter 10, “Maintaining Detail Logical Tables and Detail Tables,” on page 155
- Chapter 11, “Maintaining OLAP Groups and OLAP Summary Data Stores,” on page 175
- Chapter 12, “Maintaining Information Marts,” on page 239

Maintain a Data Warehouse

Here are some typical maintenance tasks for a data warehouse.

- Schedule Jobs to load or refresh your data warehouse. For details, see Chapter 15, “Scheduling Jobs,” on page 305.
- Use the copy feature in the SAS/Warehouse Administrator Explorer to copy a given group or data store and paste the copy under a valid parent. Use this method to model a new group or data store after an old one rather than creating a completely new object. For details about the copy feature, display the Explorer. From the menu bar, select **`Help`**, then **`Using This Window`**, then **`Explorer Pop-Up Menus`**.
- Use the Publish HTML add-in to document the metadata for a given object that is selected in the SAS/Warehouse Administrator Explorer. For details about add-ins, see “Customizing SAS/Warehouse Administrator” on page 17.
- Use the Impact Analysis add-in to trace the impact of a proposed change to a data store. For example, if you wanted to change the name of a column in an ODD, you could find out how many data stores in a Warehouse Environment include that column. For details about add-ins, see “Customizing SAS/Warehouse Administrator” on page 17.
- Use the appropriate software to back up the data and metadata for each Warehouse Environment. For details about the metadata repositories for an Environment, see “Overview: Metadata Repositories” on page 313.

- Use the Metadata Copy Wizard to copy a Warehouse Environment to a new location. You can use this feature to model a new Environment after an old one rather than creating a completely new Environment. For details, see “Metadata Copy Wizard” on page 64.

Exploit a Data Warehouse

Generally, SAS/Warehouse Administrator is used to build a data warehouse, not exploit it. Its main role is to create and manage warehouse data stores that are then accessed by other applications. In some cases, however, you can use SAS/Warehouse Administrator to prepare data or metadata for exploitation.

For example, here is one way to use SAS/EIS to analyze a summary data store.

1. Use SAS/Warehouse Administrator to export metadata from the relevant summary data store to a SAS/EIS metabase. For details about this task, see “Example: Exporting Metadata to SAS/EIS Software” on page 316.
2. In SAS/EIS, design the report you need.
3. Run the report.

Another example is to surface a copy of the metadata for a given Data Warehouse. This method could be used to support business analysts who need to answer such questions as What are the subject areas in this Data Warehouse? What data stores are available for analysis? What is the origin of the data in this column? What does this column mean? How is it derived?

1. Export metadata from the Data Warehouse. For details, see “Exporting Metadata for Groups and Data Stores” on page 316.
2. Download the MetaSpace Explorer from the SAS Web site, as described in “MetaSpace Explorer” on page 315.
3. Install and configure the MetaSpace Explorer to enable access to the exported metadata. For details about this task, see the documentation for the MetaSpace Explorer.

Customizing SAS/Warehouse Administrator

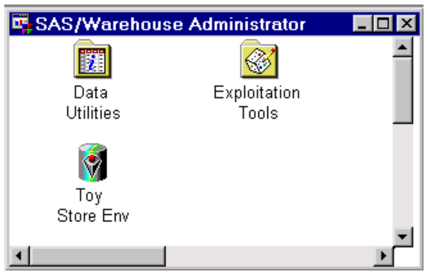
To customize SAS/Warehouse Administrator for your site, you can

- Install software from the *SAS/Warehouse Administrator Solutions* CD, which contains a number of add-ins and other tools.
- Download add-in tools from the SAS Web site.
  - Go to the SAS Web site: **`http://www.sas.com`**
  - Select **`Demos/Downloads`**
  - Select **`SAS/Warehouse Administrator Software`**
  - Select **`Add-In Tools for SAS/Warehouse Administrator 2.0`**.
- Use SAS Component Language (SCL) to write add-in tools, modify SLISTs, and make other customizations. See “Add-In Tools” on page 337 and other topics discussed in that appendix.
- Use the Metadata Application Programming Interface (API) to read or write SAS/Warehouse Administrator metadata, as described in *SAS/Warehouse Administrator 2.3 Metadata API Reference*.
- Use the Metadata API to write add-in generator applications, which you can add to the Process Library or specify directly on the Source Code tab for a process. See “Add-In Code Generator Technical Reference” on page 355.

Starting SAS/Warehouse Administrator

After SAS/Warehouse Administrator has been installed, you can start it by running SAS, typing **`dw`** on the SAS command line, and pressing the RETURN. SAS/Warehouse Administrator will open the default desktop.

Display 2.7 Desktop for SAS/Warehouse Administrator



The default desktop includes folders for Data Utilities and Exploitation Tools. The Data Utilities folder contains SAS utilities that are used to view, print, or query the contents of warehouse tables. The Exploitation Tools folder contains SAS applications that are used to exploit a data warehouse after it is built. The applications that appear in this folder will vary from site to site.

If any Warehouse Environments have been defined for your site, their icons will appear on the desktop. The Toy Store Env icon in Display 2.7 on page 18 represents the main example Environment described in this document.

If no Environments appear on your desktop, you must add one before you can work with SAS/Warehouse Administrator. You could start by adding the example Environment that was installed with SAS/Warehouse Administrator (the Marketing Warehouse Environment). For details, see Appendix 2, “Adding the Example Environment,” on page 333.

To open an Environment in the Explorer, position the cursor on an Environment icon, click the right mouse button, and select **`Edit`** from the pop-up menu. (Under Microsoft Windows and OS/2 operating environments, you can simply double-click the Environment icon.)

Keep the following in mind when you start SAS/Warehouse Administrator:

- □ If you do not specify another desktop folder in the **`dw`** command used to start SAS/Warehouse Administrator, the system uses the default SASUSER.FOLDER.SAS_WA.FOLDER. Use the FOLDER= option to specify another desktop folder, if desired.
- □ If you open a Warehouse Environment whose Data Warehouses and other elements have been defined, but these elements are not visible in the Explorer, see “Opening an Environment with a Relative Pathname” on page 63.

PART 2

Planning

CHAPTER

3

Planning Your Hardware and Software

Overview

Use this chapter to identify the hardware and software that is required for your data warehousing project. The configuration sections give examples of how SAS/Warehouse Administrator can be deployed in order to access its data sources and data stores.

General Hardware Requirements

SAS/Warehouse Administrator and its metadata repositories must be installed on a PC or a workstation that is running Microsoft Windows, Microsoft Windows NT, IBM

OS/2, or a supported version of UNIX. However, data sources, warehouse data stores, code libraries, and other warehouse resources can be stored on any host that can run any currently supported release of SAS.

Your SAS installation coordinator will help you select appropriate hardware for your project. The configuration sections in this chapter show some typical hardware configurations.

General Software Requirements

Table 3.1 on page 22 describes the SAS software that you must have in order to use SAS/Warehouse Administrator.

Table 3.1 Software Required by SAS/Warehouse Administrator

Product	Purpose	Where Installed
Base SAS	Manages SAS data sets; provides many data management functions; supports other SAS products.	Administrator's workstation(s); any warehouse host or source data host that requires Base SAS functionality or SAS product support.
SAS/Warehouse Administrator	Automates data warehouse creation and maintenance.	Administrator's workstation(s).

Note: Release 2.0 of SAS/Warehouse Administrator requires Version 8.0 or later of SAS, but data sources and warehouse data stores can be managed by any currently supported release of SAS. △

Table 3.2 on page 22 lists other software that you must have if your project requires support for the tasks described in the Purpose column.

Table 3.2 Software Required for Special Tasks

Product	Purpose	Where Installed
SAS/CONNECT	Supports connections to files, hardware resources, and SAS software on various remote hosts to use with a SAS session on a local host.	On both the local and remote hosts.
SAS/SHARE	Controls concurrent access to metadata libraries by multiple SAS/Warehouse Administrator users. Controls concurrent access to Jobs Information libraries by Jobs and SAS/Warehouse Administrator users.	Server is installed on host where the libraries to be controlled are stored. Licensed on any remote hosts that access these libraries.
SAS/ACCESS	Provides a set of individual interfaces between SAS and proprietary DBMSs.	On host(s) where DBMS data resides.
DBMS client software	Manages connections to proprietary DBMSs. Required by Version 7 and 8 SAS/ACCESS engines.	On host(s) where SAS/ACCESS LIBNAME statements are executed.
SAS/MDDB server	Creates Multidimensional databases. Required for OLAP MDDBs.	On host where OLAP MDDB is created.

Product	Purpose	Where Installed
SPDS	Scalable Performance Data Server creates, appends, and retrieves remote SAS tables faster than other methods.	Host(s) where remote SAS tables reside.
SAS/AF	SAS application development environment. Required to modify or write SAS/Warehouse Administrator metadata API applications. See Note below about Information Mart Items.	Metadata API development host(s). See Note below about Information Mart Items.
SAS/GRAPH	SAS graphics support. Required to modify or write add-in software that includes a GUI.	Add-in software development host(s).
Other SAS products	Exploit warehouse data.	Depends on the product.

Note: If you create any Information Mart Items, you must either install SAS/AF on the host where the Information Mart Item is located, or replace the default “open” command with a command that does not require SAS/AF software. For details, see “Example: Creating an Information Mart Item” on page 242. △

Your SAS installation coordinator will help you select appropriate software for your project. The configuration sections in this chapter show some typical software configurations.

Metadata Host Configuration

Overview

SAS/Warehouse Administrator stores its metadata in at least two SAS libraries:

libref _MASTER	is the Data Warehouse Environment repository. It stores metadata for any data stores defined at the Environment level, including host definitions and most other global metadata. _MASTER contains references to any Data Warehouse repositories (libref _DWMD).
libref _DWMD	is a Data Warehouse repository. It stores metadata for any data stores defined at the Data Warehouse level.

Warehouse Environments and Data Warehouses are described later in this document. For now, assume they are metadata records that specify the location of the metadata repositories for a data warehousing project.

SAS/Warehouse Administrator and its metadata repositories can be installed on the same host or on different hosts. Your SAS installation coordinator will help you choose the best metadata host configuration for your project. This section describes some recommendations and typical configurations.

Recommendations

A local Warehouse Environment is an Environment whose metadata repository is stored on the SAS/Warehouse Administrator host. Because SAS/Warehouse Administrator continuously accesses this metadata, you will get better performance if this metadata is local. SAS/Warehouse Administrator response time might be slower if it has to access metadata across a remote connection.

A remote Warehouse Environment might be appropriate if you require concurrent read/write access to Warehouse Environments by multiple SAS/Warehouse Administrator hosts. In that case, you should create a remote Warehouse Environment and put its metadata repository under the control of a SAS/SHARE server remote to the SAS/Warehouse Administrator hosts. Figure 3.3 on page 26 shows an example of such a configuration.

CAUTION:

Concurrent read/write access to a SAS/Warehouse Administrator metadata repository by multiple SAS/Warehouse Administrator hosts must be controlled through a SAS/SHARE server. Otherwise, metadata can be corrupted if two or more users update the metadata at the same time. △

Metadata Configuration to Avoid

In general, avoid configurations where the SAS/Warehouse Administrator host has one machine architecture (such as Microsoft Windows NT), and the metadata host has a different architecture (such as UNIX).

Remote access to Environment metadata is managed through SAS/SHARE software, and this application does not permit access to SAS catalogs if the local and remote hosts have different machine architectures. This means, for example, that you would not be able to store Notes for tables, columns, or other objects.

Jobs Information Libraries and SAS/SHARE Software

If you want SAS/Warehouse Administrator to generate code that will execute a Job at a future date and time, you must define one or more Jobs Information libraries. A Jobs Information library is a SAS library that contains status information for Jobs that have been scheduled through the Job Properties window in SAS/Warehouse Administrator. If job tracking is enabled for a given Job, when the Job executes, it will update its status in the appropriate Jobs Information library.

Jobs Information libraries are separate from the metadata repositories for an Environment or a Data Warehouse. A Warehouse Environment and its Data Warehouses can share one Jobs Information library, or they could have separate Jobs Information libraries.

CAUTION:

For a production data warehouse in which Jobs are scheduled and tracked through SAS/Warehouse Administrator, it is strongly recommended that you place the Jobs Information libraries under the control of a SAS/SHARE server. △

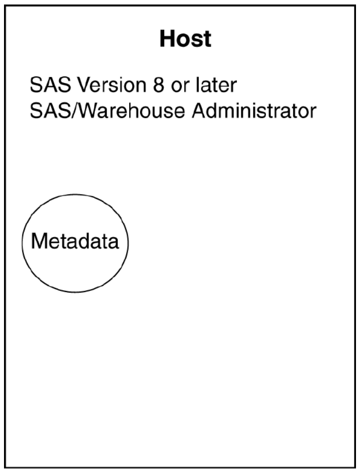
A SAS/SHARE server will prevent sharing conflicts between running Jobs with tracking options enabled and SAS/Warehouse Administrator users who want to schedule Jobs or view the status of Jobs.

For a non-production data warehouse or for a data warehouse where Jobs are not tracked through SAS/Warehouse Administrator, it is not necessary to put the relevant Jobs Information libraries under the control of a SAS/SHARE server. For details about when a Job is tracked in SAS/Warehouse Administrator, see “Tracking Jobs” on page 307.

Local Metadata: Single Host

In the single host configuration, both SAS/Warehouse Administrator and its metadata are installed on the same machine.

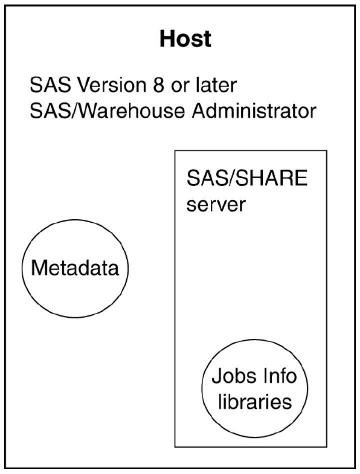
Figure 3.1 Single Host



Support for Job Scheduling and Tracking

Figure 3.2 on page 25 shows the same configuration with additional software to support the scheduling and tracking of Jobs through SAS/Warehouse Administrator. Note that a SAS/SHARE server controls access to the Jobs Information libraries.

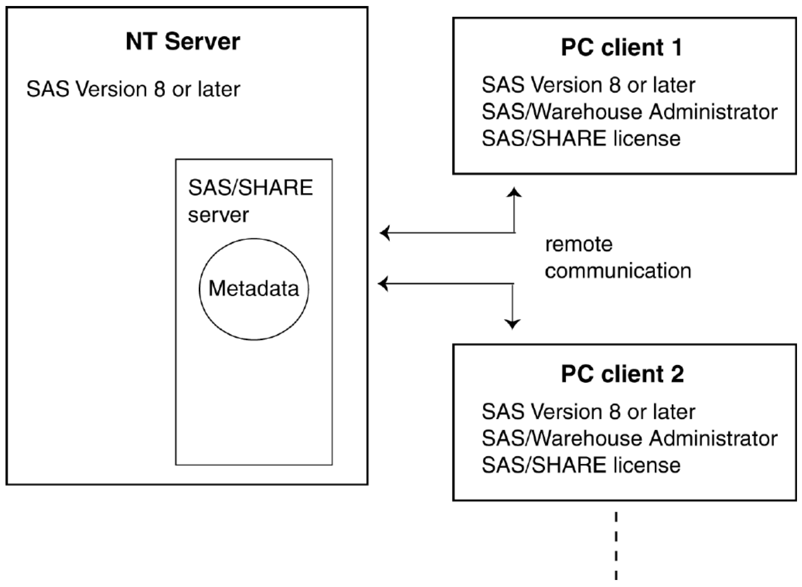
Figure 3.2 Single Host with Support for Job Scheduling and Tracking



Remote Metadata: PC Client to Windows NT Server

This configuration is one possible solution for projects that require concurrent read/write access to the metadata repositories by multiple SAS/Warehouse Administrator users.

Figure 3.3 PC Client to Windows NT Server

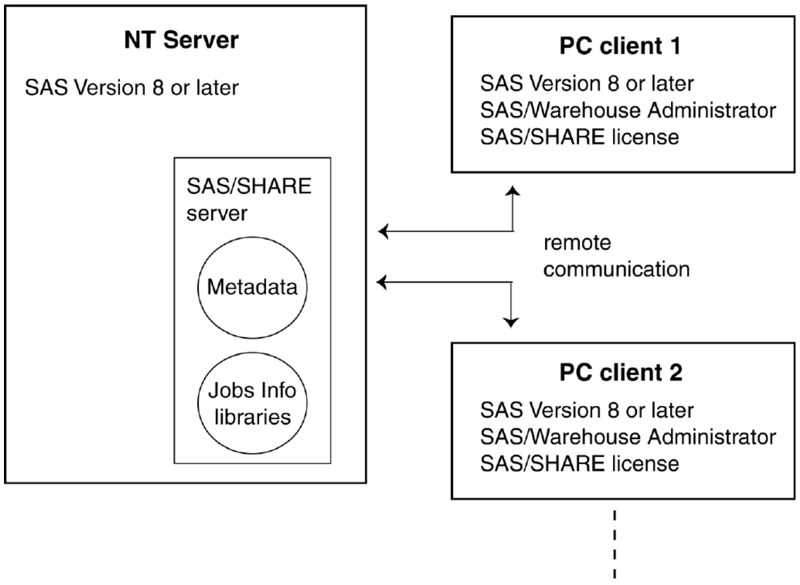


In this configuration, SAS/Warehouse Administrator is installed on one or more PC clients, and SAS/Warehouse Administrator metadata is installed on a remote machine running the Windows NT Server operating system. A SAS/SHARE server on the Windows NT host handles remote communications to and from the SAS/Warehouse Administrator clients. Each SAS/Warehouse Administrator client must have a license for SAS/SHARE, although this application does not have to be installed on each client.

Support for Job Scheduling and Tracking

Figure 3.4 on page 27 shows the same configuration with additional software to support the scheduling and tracking of Jobs through SAS/Warehouse Administrator. Note that a SAS/SHARE server controls access to the Jobs Information libraries.

Figure 3.4 PC Client to Windows NT Server with Support for Job Scheduling and Tracking



Remote Metadata: UNIX or NT Server to Like Server

A server-to-like-server configuration—an NT Server to an NT Server, or a UNIX host to a UNIX host—is also possible. For example, in a UNIX configuration, SAS/ Warehouse Administrator is installed on one or more UNIX hosts, and its metadata is installed on another UNIX host. This configuration is another solution for projects that require concurrent read/write access to the metadata repositories by multiple SAS/ Warehouse Administrator users.

Excluding the operating systems, the software that is required to implement this configuration is similar to that described in "Remote Metadata: PC Client to Windows NT Server" on page 25.

Data Host Configuration

Overview

Data sources and warehouse data stores can reside on any host that can run any currently supported release of SAS. Your SAS installation coordinator will help you choose the best data host configuration for your project. Some typical configurations are described in this section.

Note: For simplicity, the figures in this section show SAS/Warehouse Administrator and its metadata repositories installed on the same host. This is not a requirement. Any of the metadata host configurations described in "Metadata Host Configuration" on page 23 could also be used with the data host configurations described in this section. △

Access to Data In DBMS Format

SAS and SAS/Warehouse Administrator can be used to read or write data in a proprietary data base management system (DBMS), such as DB2 or Oracle. Under Version 7 or later of SAS, you can use a SAS/ACCESS LIBNAME statement to read and write DBMS data as if it were SAS data. Under Version 6 or later of SAS, you can use other SAS/ACCESS methods, such as an SQL Pass-through view to read DBMS data, and PROC DBLOAD to load data into DBMS tables.

If your project plan requires you to read or write DBMS data, your data host configuration must include the software to support at least one of the methods in the following two tables.

Note: By default, for new DBMS data stores, SAS/Warehouse Administrator generates code that uses SAS/ACCESS LIBNAME statements to read and write data in DBMS format. △

Table 3.3 SAS/ACCESS LIBNAME Method

Product	Purpose	Where Installed
Base SAS, Version 7 or later	Supports SAS/ACCESS LIBNAME statement.	On the host where the SAS/ ACCESS LIBNAME statement is executed.
SAS/ACCESS engine for target DBMS	Read or write DBMS data as if it were SAS data.	same
Client software for target DBMS	Handles local or remote connection to DBMS.	same

Note: For DBMS tables created with 1.x releases of SAS/Warehouse Administrator, or for new DBMS tables with code generation level 1.1 Load Steps, SAS/Warehouse Administrator creates an SQL Pass-through view to access DBMS data and uses PROC DBLOAD to load data into DBMS tables. △

Table 3.4 Other SAS/ACCESS Methods

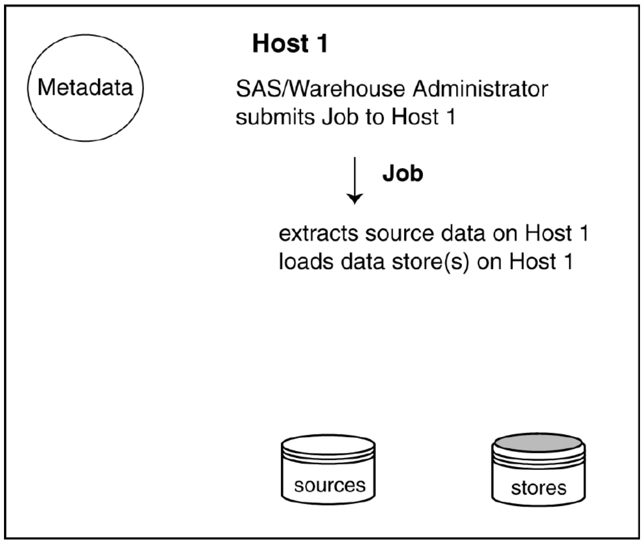
Product	Purpose	Where Installed
Base SAS, Version 6 or later	Supports SAS/ACCESS	On the host where the SAS/ ACCESS statement is executed.
SAS/ACCESS	Read or write DBMS data.	same
SAS/CONNECT	Supports connections to files, hardware resources, and SAS software on various remote hosts to use with a SAS session on a local host.	On both the local and remote hosts.

A third alternative is to use SAS/Warehouse Administrator add-ins that generate the code required to load a warehouse table into a DBMS, such as Oracle, using native loading software. For instructions on obtaining add-ins, see “Customizing SAS/ Warehouse Administrator” on page 17.

Local Job, Local Data

In this configuration, SAS/Warehouse Administrator, its data sources, and its warehouse data stores are installed on the same machine. The Job and all of its processes execute on the SAS/Warehouse Administrator host.

Figure 3.5 Local Job, Local Data


Host 1
Metadata
SAS/Warehouse Administrator submits Job to Host 1
Job
extracts source data on Host 1
loads data store(s) on Host 1
sources
stores

The configuration shown in Figure 3.5 supports the following scenario:

1 SAS/Warehouse Administrator submits the Job to its SAS session on Host 1.
2 The Job calls one or more extraction processes that execute on Host 1 and send their output to Host 1.
3 The Job calls one or more Load processes that execute on Host 1 and send their output to Host 1.

To implement a data host configuration similar to the one shown in Figure 3.5 on page 29, you will need the following software:

Table 3.5 Required Software: Local Job, Local Data

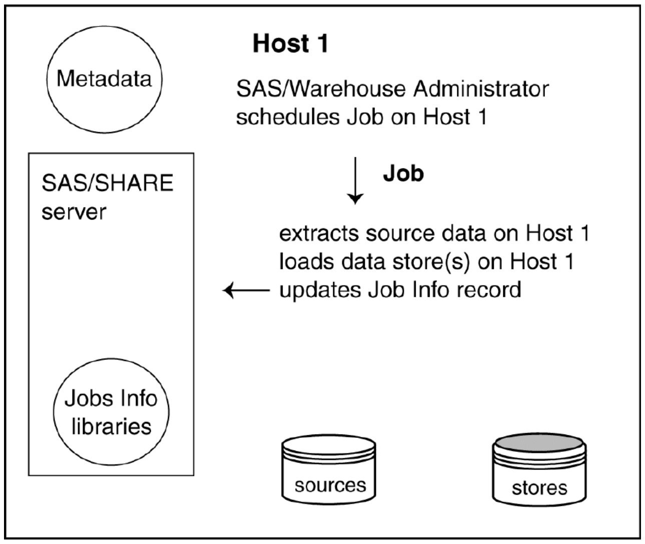
Software	Host
Base SAS, Version 8 or later	Host 1
SAS/Warehouse Administrator	Host 1

If your project plan requires you to read or write DBMS data, your configuration must include the software to support at least one of the methods described in “Access to Data In DBMS Format” on page 28.

Support for Job Scheduling and Tracking

Figure 3.6 on page 30 shows the same configuration with additional software to support the scheduling and tracking of Jobs through SAS/Warehouse Administrator. Note that a SAS/SHARE server controls access to the Jobs Information libraries.

Figure 3.6 Local Job, Local Data with Support for Job Scheduling and Tracking


Host 1
Metadata
SAS/Warehouse Administrator schedules Job on Host 1
SAS/SHARE server
Job
extracts source data on Host 1
loads data store(s) on Host 1
updates Job Info record
Jobs Info libraries
sources
stores

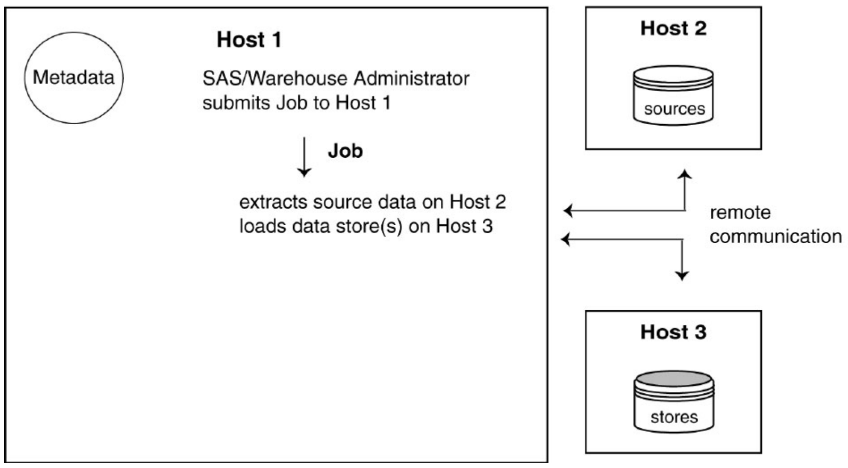
The configuration shown in Figure 3.6 supports the following scenario:

1 SAS/Warehouse Administrator submits the Job to a scheduling server application on Host 1, such as CRON on a UNIX host, or the AT command (Schedule service) on a Windows NT host.
2 The Job calls one or more extraction processes that execute on Host 1 and send their output to Host 1.
3 The Job calls one or more Load processes that execute on Host 1 and send their output to Host 1.
4 The Job updates the Jobs Information library with the status of the Job.

Local Job, Remote Data

In this configuration, SAS/Warehouse Administrator is installed on one host, and its data sources and data stores reside on one or more remote hosts. The Job executes on the SAS/Warehouse Administrator host. The processes within the Job can execute on the same host, or they can be submitted remotely from the Job to run on the remote hosts.

Figure 3.7 Local Job, Remote Data


Host 1
Metadata
SAS/Warehouse Administrator submits Job to Host 1
Job
extracts source data on Host 2
loads data store(s) on Host 3
Host 2
sources
remote communication
Host 3
stores

The configuration shown in Figure 3.7 supports the following scenario:

1. SAS/Warehouse Administrator submits the Job to its SAS session on Host 1.
2. The Job calls one or more extraction processes that execute on Host 1, read data from Host 2, and send their output to an appropriate host.
3. The Job calls one or more Load processes that execute on Host 1 and send their output to Host 3.

To implement a data host configuration similar to the one shown in Figure 3.7 on page 31, you will need the following software:

Table 3.6 Required Software: Local Job, Remote Data

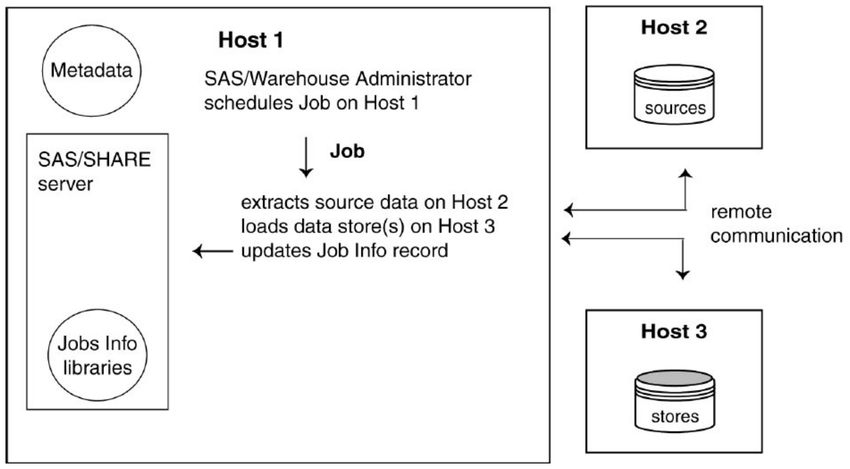
Software	Host
Base SAS, Version 8 or later	Host 1
Base SAS, any supported version	Host 2, Host 3
SAS/Warehouse Administrator	Host 1
Remote communication software	See Note below

Note: In the data host configuration shown in Figure 3.7 on page 31, if the processes on Host 1 must read or write SAS data on a remote host, SAS/CONNECT is required on Host 1 and the remote host. If the processes on Host 1 must read from or write to a proprietary DBMS, your configuration must include the software to support at least one of the methods described in “Access to Data In DBMS Format” on page 28. △

Support for Job Scheduling and Tracking

Figure 3.8 on page 32 shows the same configuration with additional software to support the scheduling and tracking of Jobs through SAS/Warehouse Administrator. Note that a SAS/SHARE server controls access to the Jobs Information libraries.

Figure 3.8 Local Job, Remote Data with Support for Job Scheduling and Tracking



The configuration shown in Figure 3.8 supports the following scenario:

1 SAS/Warehouse Administrator submits the Job to a scheduling server application on Host 1.

2 The Job calls one or more extraction processes that execute on Host 1, read data from Host 2, and send their output to an appropriate host.

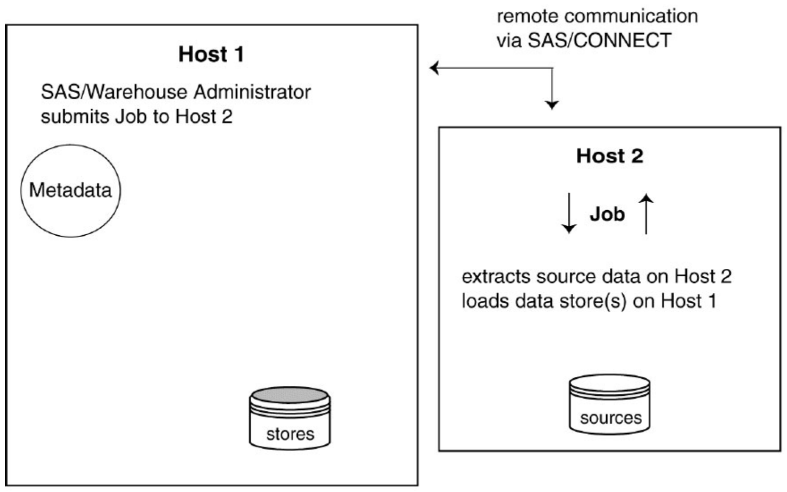
3 The Job calls one or more Load processes that execute on Host 1 and send their output to Host 3.

4 The Job updates the Jobs Information library with the status of the Job.

Remote Job, Local Data

In this configuration, SAS/Warehouse Administrator is installed on one host; its data sources reside on one or more remote hosts, and its data stores reside on the SAS/Warehouse Administrator host. The Job is submitted to a remote host, and all of its processes execute on remote host(s), where the source data is located.

Figure 3.9 Remote Job, Local Data



The configuration shown in Figure 3.9 supports the following scenario:

1 SAS/Warehouse Administrator submits the Job to a SAS session on Host 2.

2 The Job calls one or more extraction processes that execute on Host 2, read data from Host 2, and send their output to an appropriate host.

3 The Job calls one or more Load processes that execute on Host 2 and use PROC UPLOAD to send their output to Host 1.

To implement a data host configuration similar to the one shown in Figure 3.9 on page 33, you will need the following software:

Table 3.7 Required Software: Remote Job, Local Data

Software	**Host**
Base SAS, Version 8 or later	Host 1
Base SAS, version appropriate for the processes being executed	Host 2
SAS/Warehouse Administrator	Host 1
SAS/CONNECT	Host 1, Host 2

SAS/CONNECT is required on the SAS/Warehouse Administrator host and the Job host in order to support Remote Compute Services.

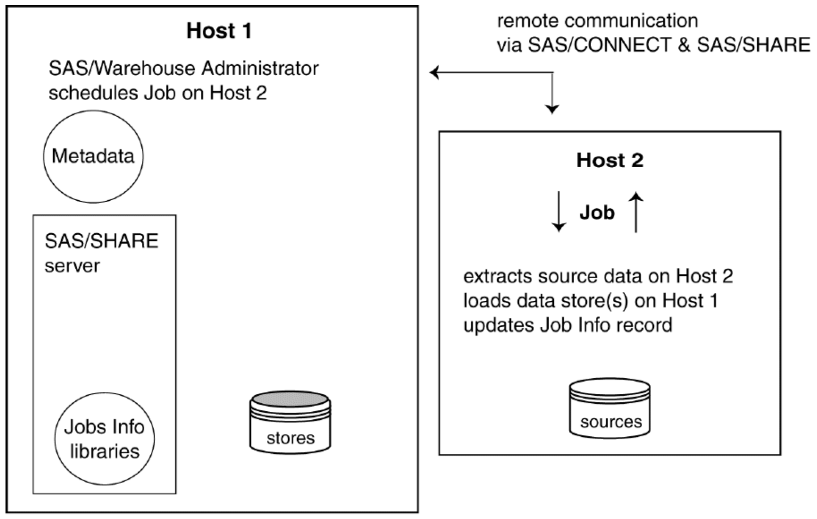
If the processes on Host 2 need to read from or write to a proprietary DBMS, your configuration must include the software to support at least one of the methods described in “Access to Data In DBMS Format” on page 28.

Support for Job Scheduling and Tracking

Figure 3.10 on page 34 shows the same configuration with additional software to support the scheduling and tracking of Jobs through SAS/Warehouse Administrator.

Note that a SAS/SHARE server controls access to the Jobs Information libraries on Host 1. The Job on Host 2 uses SAS/SHARE to communicate with the SAS/SHARE server. Accordingly, Host 2 must have a license for SAS/SHARE.

Figure 3.10 Remote Job, Local Data with Support for Job Scheduling and Tracking


Host 1
SAS/Warehouse Administrator schedules Job on Host 2
Metadata
SAS/SHARE server
Jobs Info libraries
stores
remote communication via SAS/CONNECT & SAS/SHARE
Host 2
Job
extracts source data on Host 2
loads data store(s) on Host 1
updates Job Info record
sources

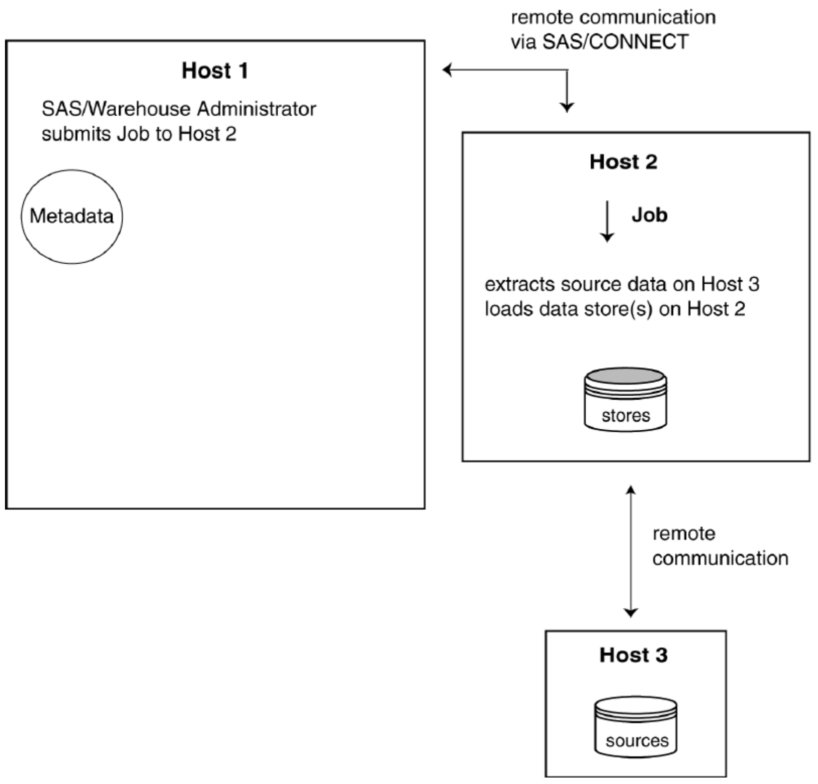
The configuration shown in Figure 3.10 supports the following scenario:

1 SAS/Warehouse Administrator submits the Job to a scheduling server application on Host 2.
2 The Job calls one or more extraction processes that execute on Host 2, read data from Host 2, and send their output to an appropriate host.
3 The Job calls one or more Load processes that execute on Host 2 and send their output to Host 1.
4 The Job updates the Jobs Information library with the status of the Job.

Remote Job, Remote Data (Example 1)

In this configuration, SAS/Warehouse Administrator is installed on one host; its data sources reside on one or more remote hosts, and its data stores reside on one or more remote hosts. The Job is submitted to a remote host, and all of its processes execute on remote host(s), where the source data and the data stores are located. You could use this configuration to execute the SAS/Warehouse Administrator Job on a remote host where data stores are located.

Figure 3.11 Remote Job, Remote Data (Example 1)


remote communication via SAS/CONNECT
Host 1
SAS/Warehouse Administrator submits Job to Host 2
Metadata
Host 2
Job
extracts source data on Host 3
loads data store(s) on Host 2
stores
remote communication
Host 3
sources

The configuration shown in Figure 3.11 supports the following scenario:

1 SAS/Warehouse Administrator submits the Job to a SAS session on Host 2.

2 The Job calls one or more extraction processes that execute on Host 2, read data from Host 3, and send their output to an appropriate host (probably Host 3).

3 The Job calls one or more Load processes that execute on Host 2 and send their output to Host 2.

To implement a data host configuration similar to the one shown in Figure 3.11 on page 35, you will need the following software:

Table 3.8 Required Software: Remote Job, Remote Data (Example 1)

Software	**Host**
Base SAS, Version 8 or later	Host 1
Base SAS, version appropriate for the processes being executed	Host 2
Base SAS, any supported version	Host 3
SAS/Warehouse Administrator	Host 1
SAS/CONNECT	Host 1, Host 2, perhaps Host 3

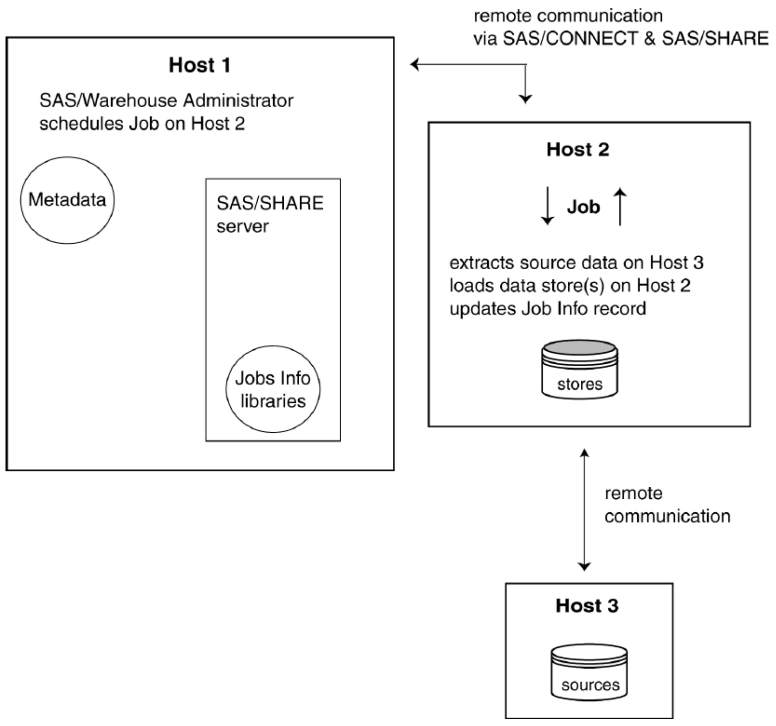
SAS/CONNECT is required on the SAS/Warehouse Administrator host (Host 1) and the Job host (Host 2) in order to support Remote Compute Services.

If the extraction processes on Host 2 need to read SAS data on a remote host (Host 3), SAS/CONNECT or SAS/SHARE is required on Host 2 and the remote host. If the processes on Host 2 needs to read from or write to a proprietary DBMS on a remote host, your configuration must include the software to support at least one of the methods described in “Access to Data In DBMS Format” on page 28.

Support for Job Scheduling and Tracking

Figure 3.12 on page 36 shows the same configuration with additional software to support the scheduling and tracking of Jobs through SAS/Warehouse Administrator. Note that a SAS/SHARE server controls access to the Jobs Information libraries on Host 1. The Job on Host 2 uses SAS/SHARE to communicate with the SAS/SHARE server. Accordingly, Host 2 must have a license for SAS/SHARE.

Figure 3.12 Example 1 with Support for Job Scheduling and Tracking



The configuration shown in Figure 3.12 supports the following scenario:

1 SAS/Warehouse Administrator submits the Job to a scheduling server application on Host 2.

2 The Job calls one or more extraction processes that execute on Host 2, read data from Host 3, and send their output to an appropriate host (probably Host 3).

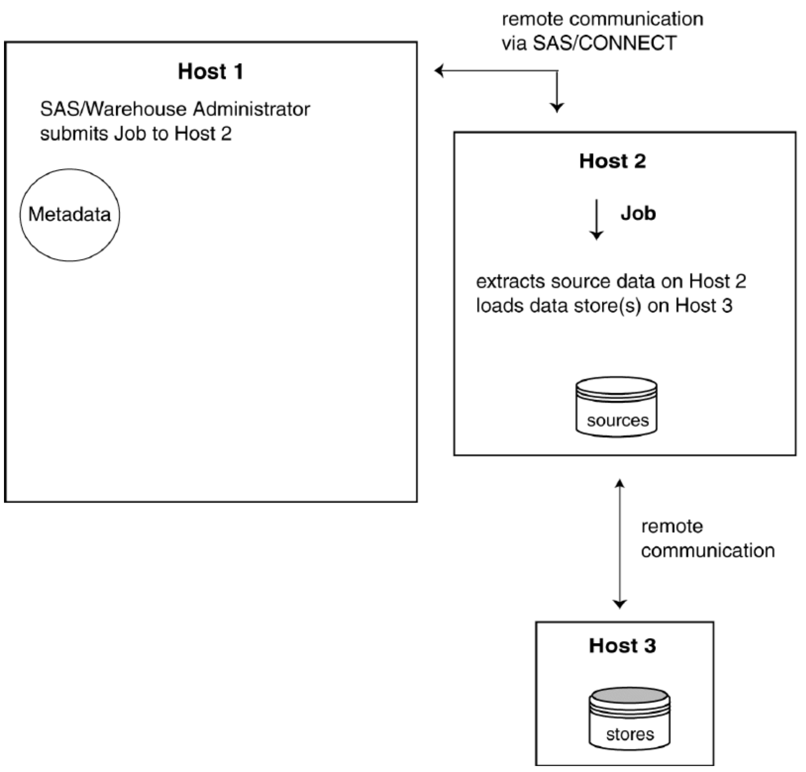
3 The Job calls one or more Load processes that execute on Host 2 and send their output to Host 2.

4 The Job updates the Job Information library with the status of the Job.

Remote Job, Remote Data (Example 2)

This configuration is similar to the one described in “Remote Job, Remote Data (Example 1)” on page 34. You could use this configuration to execute the SAS/Warehouse Administrator Job on a remote host where data sources are located.

Figure 3.13 Remote Job, Remote Data, Example 2


remote communication via SAS/CONNECT
Host 1
SAS/Warehouse Administrator submits Job to Host 2
Metadata
Host 2
Job
extracts source data on Host 2 loads data store(s) on Host 3
sources
remote communication
Host 3
stores

The configuration shown in Figure 3.13 supports the following scenario:

1 SAS/Warehouse Administrator submits the Job to a SAS session on Host 2.

2 The Job calls one or more extraction processes that execute on Host 2, read data from Host 2, and send their output to an appropriate host.

3 The Job calls one or more Load processes that execute on Host 2 and send their output to Host 3.

To implement a data host configuration similar to the one shown in Figure 3.13 on page 37, you will need the following software:

Table 3.9 Required Software: Remote Job, Remote Data (Example 2)

Software	**Host**
Base SAS, Version 8 or later	Host 1
Base SAS, version appropriate for the processes being executed	Host 2
Base SAS, any supported version	Host 3

Software	Host
SAS/Warehouse Administrator	Host 1
SAS/CONNECT	Host 1, Host 2, perhaps Host 3

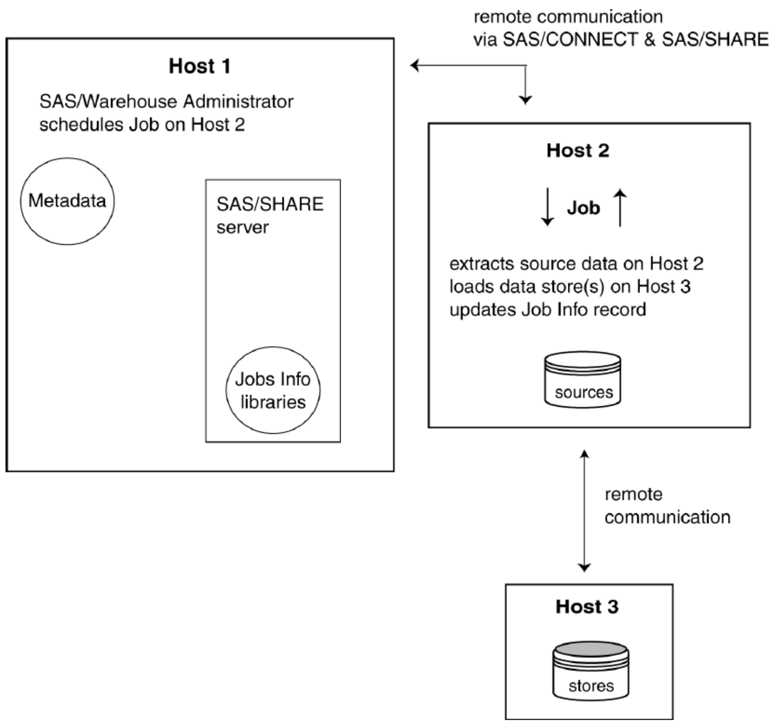
SAS/CONNECT is required on the SAS/Warehouse Administrator host (Host 1) and the Job host (Host 2) in order to support Remote Compute Services.

If the Load processes on Host 2 need to write SAS data on a remote host, SAS/CONNECT or SAS/SHARE is required on Host 2 and the remote host (Host 3). If the processes on Host 2 need to read from or write to a proprietary DBMS on a remote host, your configuration must include the software to support at least one of the methods described in “Access to Data In DBMS Format” on page 28.

Support for Job Scheduling and Tracking

Figure 3.14 on page 38 shows the same configuration with additional software to support the scheduling and tracking of Jobs through SAS/Warehouse Administrator. Note that a SAS/SHARE server controls access to the Jobs Information libraries on Host 1. The Job on Host 2 uses SAS/SHARE to communicate with the SAS/SHARE server. Accordingly, Host 2 must have a license for SAS/SHARE.

Figure 3.14 Example 2 with Support for Job Scheduling and Tracking



The configuration shown in Figure 3.14 supports the following scenario:

1 SAS/Warehouse Administrator submits the Job to a scheduling server application on Host 2.

2 The Job calls one or more extraction processes that execute on Host 2, read data from Host 2, and send their output to an appropriate host.
3 The Job calls one or more Load processes that execute on Host 2 and send their output to Host 3.
4 The Job updates the Jobs Information library with the status of the Job.

What's Next

After you plan your hardware and software configuration, you are ready to determine the SAS/Warehouse Administrator objects and processes that you will need for your data warehousing project.

CHAPTER 4

Planning Your Data Stores and Processes

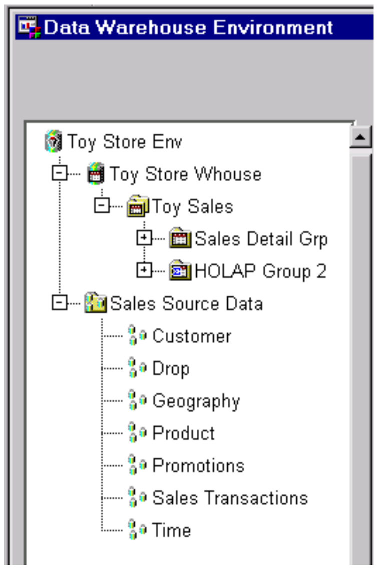
Overview

Use this chapter to identify the groups, data stores, and processes that you need to implement the main data collections in your data warehousing project. For example, suppose you had identified a set of sales data that you wanted to manage in a data warehouse. You could use this chapter to identify the SAS/Warehouse Administrator elements required to access this sales data and store it in a data warehouse. You might want to plan and implement one data collection at a time.

Groups and Data Stores

For each data warehousing project, you will create a hierarchy of groups and data stores in the SAS/Warehouse Administrator Explorer, such as the one shown in Display 4.1 on page 42.

Display 4.1 Warehouse Environment in the Explorer

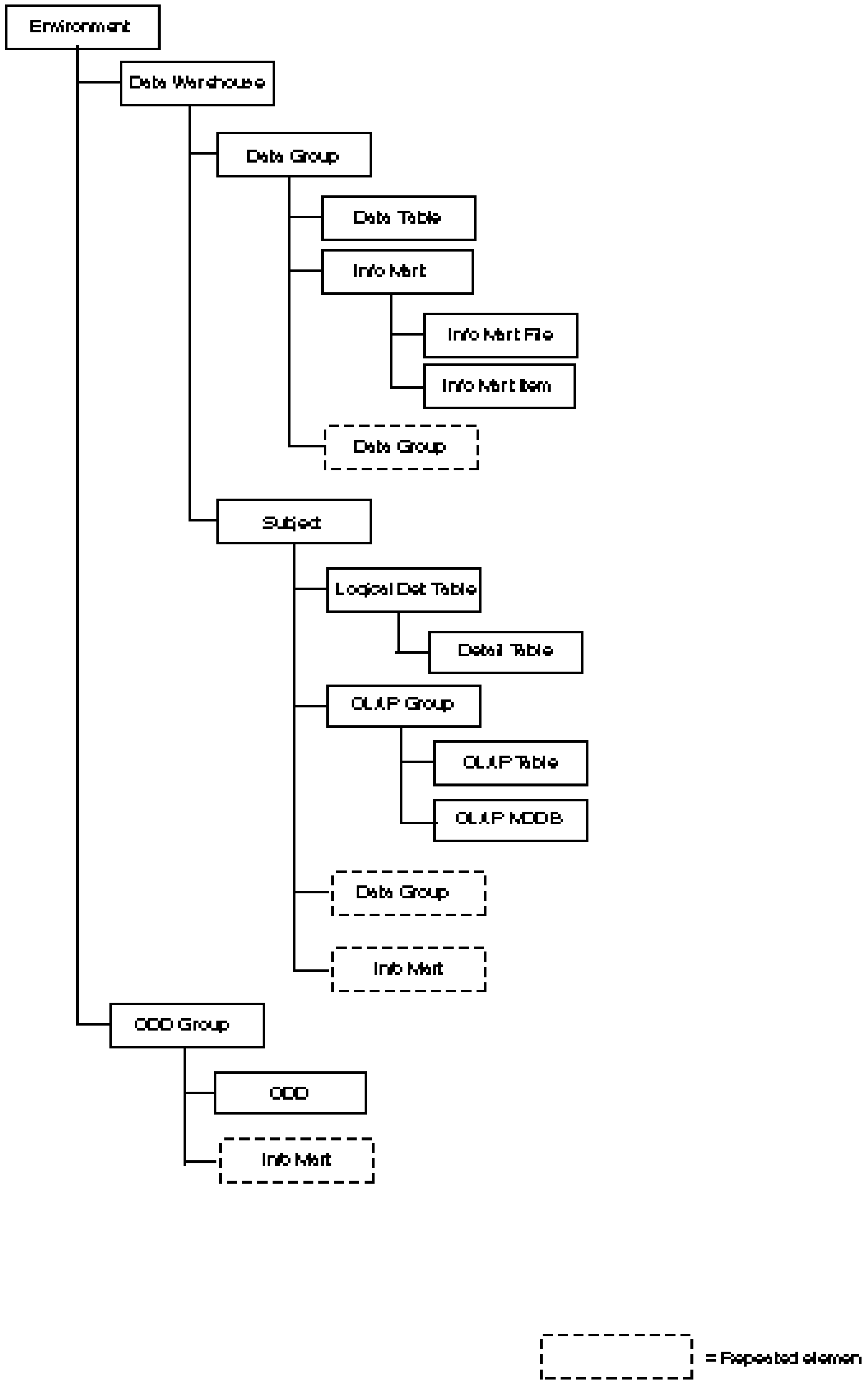


A group is an element in the Explorer or the Process Editor that is used to organize other elements. For example, in Display 4.1 on page 42, Toy Store Env is the top-level group, and Toy Store Whouse and Sales Source Data are subgroups.

A data store is a table, a view, or a file that is registered in a Warehouse Environment or in one of its Data Warehouses. For example, the Customer item in Display 4.1 on page 42 is a data store that contains source data about toy customers.

The Explorer enforces a certain hierarchy of groups and data stores. You can only add a Data Warehouse object to a Warehouse Environment object, for example. Figure 4.1 on page 43 illustrates the hierarchy of objects in the Explorer.

Figure 4.1 Hierarchy of Groups and Data Stores in the Explorer



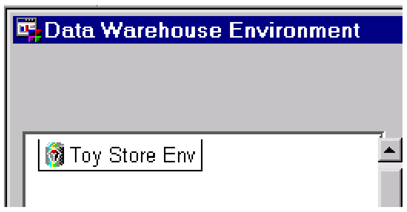
The sections that follow provide details about each kind of group and data store.

Data Warehouse Environments

Each data warehousing project requires at least one Data Warehouse Environment. A Data Warehouse Environment is 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.

Environments are added and opened from the SAS/Warehouse Administrator desktop, as described in "Opening a Warehouse Environment in the Explorer" on page 63. Display 4.2 on page 44 shows a new Environment (Toy Store Env) that has been opened in the SAS/Warehouse Administrator Explorer.

Display 4.2 New Warehouse Environment in the Explorer



Display 4.1 on page 42 shows an Environment in which a number of subgroups and data stores have been defined. For details, see Chapter 5, "Maintaining Environments," on page 61.

Global Metadata

After you open a Warehouse Environment in the Explorer, you can define metadata records that can be shared by data stores and other elements within that Environment. For example, if a number of data stores in an Environment will be stored on the same host, you could create a definition for that host and save it as part of the global metadata for the Environment. You could then include the host definition in the metadata for the data stores that reside on that host, without retyping the host information each time.

In each Warehouse Environment, you will define at least some of the global metadata types below:

- SAS library definitions (including SAS/ACCESS LIBNAME definitions and Jobs Information library definitions)
- Host definitions
- DBMS connection profiles
- Contact records
- Scheduling server definitions

As you create data stores and other SAS/Warehouse Administrator elements, you must specify hosts, libraries, and other global metadata. If you create the main global metadata items first, you can simply select them from a list, rather than having to stop and create a host definition in the middle of creating a data store. For details, see Chapter 6, "Maintaining Global Metadata," on page 75.

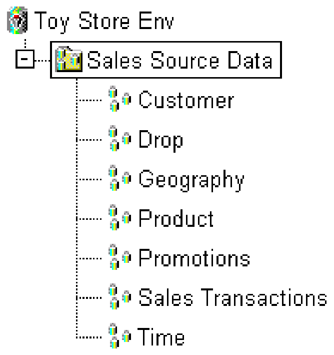
Operational Data Definitions (ODDs)

After you have created an Environment, you can register the source data for that Environment. To do that, you must define at least one Operational Data Definition Group (ODD Group) and a number of Operational Data Definitions (ODDs).

An ODD Group is a simple grouping element for ODDs. It can also contain one or more Information Marts, another kind of SAS/Warehouse Administrator group. In the SAS/Warehouse Administrator Explorer, an ODD Group can only be added to a

Warehouse Environment. Display 4.3 on page 45 shows one ODD Group (Sales Source Data) that contains a number of ODDs (Customer, Drop, and so on).

Display 4.3 ODD Group with ODDs in the Explorer



An Operational Data Definition (ODD) is a metadata record that provides access to data sources. The ODDs, in turn, are used as inputs to data stores in a Warehouse Environment.

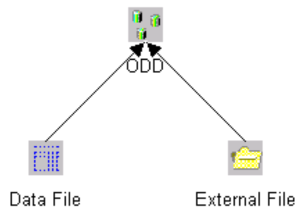
At a minimum, in order for a data source to be visible in a Warehouse Environment, you must specify the location of that data source in an ODD. You can define an ODD that simply registers the location of a SAS table or view, or that registers the location of a DBMS table with the help of a SAS/ACCESS 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, an ODD can be added only to an ODD Group. For details about ODDs, see Chapter 7, “Registering Data Sources,” on page 107.

Data Files and External Files

Data Files and External Files are inputs to ODDs. They cannot be added in the Explorer; they can only be added in the Process Editor. Display 4.4 on page 45 shows a Process Flow that includes a Data File and an External File.

Display 4.4 Data File and External File in a Process Flow



A Data File is a metadata record that specifies a SAS file that is an input to an ODD. You can define a Data File 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 SAS/ACCESS 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.

An External File is 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.

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 previously discussed conditions, you might want to specify a Process Flow for this Job for documentation purposes.

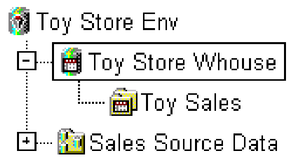
For details about these objects, see Chapter 7, "Registering Data Sources," on page 107.

Data Warehouses and Subjects

To support your data warehousing project, you will create at least one Data Warehouse and one or more Subjects within each Warehouse.

A Data Warehouse is a metadata record that specifies the SAS library _DWMD. The _DWMD library is the metadata repository for most of the groups and data stores in a data warehouse. In the Explorer, a Data Warehouse object can only be added to a Data Warehouse Environment. Display 4.5 on page 46 shows one Data Warehouse (Toy Store Whouse) that contains one Subject (Toy Sales).

Display 4.5 Data Warehouse and Subject in the Explorer



A Subject is a grouping element for data related to one topic within a Data Warehouse. For example, a Data Warehouse might have a Subject called Products (information related to products) or Sales (information related to sales). Each Subject can be composed of a number of different data collections: detail data, summary data, charts, reports, and graphs. In the Explorer, a Subject can only be added to a Data Warehouse.

For details about Data Warehouses and Subjects, see Chapter 8, "Maintaining Data Warehouses and Subjects," on page 133.

Data Marts

A data mart is a limited data warehouse that is often designed to meet the needs of a particular department or individual. A data mart is more limited in scope than a data warehouse, which typically contains information used by more than one department. To implement a data mart in SAS/Warehouse Administrator, use an appropriate SAS/Warehouse Administrator object: either a Data Warehouse or a Data Group.

Detail Data Stores

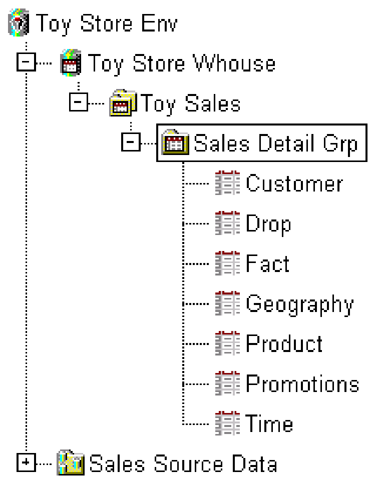
Detail data is information that is at or near the fact level in a database. It is data that has not been summarized or has only been lightly summarized after extraction from a source. In a data warehousing project, detail data stores are often used as inputs to summary data stores. They can also be exploited directly — data mining operations are typically run against the detail data in a warehouse, for example.

In SAS/Warehouse Administrator, detail data can be stored in Data Tables, Detail Logical Tables, or Detail Tables. Typically, you will create these objects whenever the source data specified in an ODD needs to be transformed or merged in order to provide useful detail data for your project.

A Data Table is 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 included as part of a join. They can be used 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, a Data Table can only be added to a Data Group. For more information about Data Tables, see Chapter 9, "Maintaining Data Tables," on page 145.

A Detail Logical Table is 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. Display 4.6 on page 47 shows a Detail Logical Table (Sales Detail Grp) with Detail Tables (Customer, Drop, and so on).

Display 4.6 Detail Logical Table with Detail Tables in the Explorer



In the SAS/Warehouse Administrator Explorer, a Detail Logical Table can only be added 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.

A Detail Table is 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, a Detail Table can be added only to a Detail Logical Table. For more information, see Chapter 10, "Maintaining Detail Logical Tables and Detail Tables," on page 155.

Which Detail Data Store Do I Need?

Typically, you will create Data Tables, Detail Logical Tables, or Detail Tables whenever the source data specified in an ODD needs to be transformed or merged in order to provide useful detail data for your project.

A Data Table is a good choice when:

- You want to create multiple groups of detail data stores in the same Subject. (A Subject can only have one Detail Logical Table.)

- □ You need an intermediate data store, such as a look-up table included as part of a join in the Process Editor. For details, see "Example: Creating a Data Table" on page 148.

A Detail Logical Table and its Detail Tables are a good choice when you want to implement a view on multiple, related Detail Tables. For more information, see "Example: Creating a Detail Logical Table as a View to Multiple Detail Tables" on page 165.

Summary Data Stores

Summary data is information that is derived from the facts in a database. It is data that has been summarized after extraction from a source. Many data warehousing projects require a number of summary data stores in order to support end-user reports, queries, and analysis.

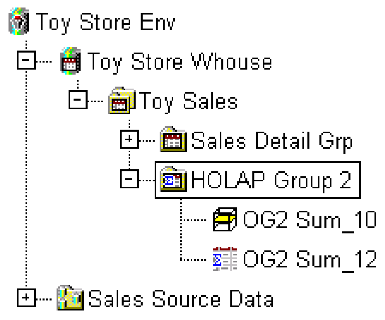
In SAS/Warehouse Administrator, most summary data is stored in *OLAP Tables* and *OLAP MDDBs*. It is possible to store summarized data in Data Tables, but you must specify user-written summarization code in the Load Step for the Data Table.

Note: SAS/Warehouse Administrator 2.0 replaces Summary Groups, Summary Tables, and Summary MDDBs with OLAP Groups, OLAP Tables, and OLAP MDDBs. That is, in Release 2.0, you cannot create Summary Groups. Instead, you can create OLAP Groups. The new summary data stores better support OLAP (online analytical processing) and facilitate the different requirements of OLAP reporting for HOLAP (hybrid OLAP), MOLAP (multidimensional OLAP), and ROLAP (relational OLAP). △

An 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 (Relational OLAP), MOLAP (Multidimensional OLAP), HOLAP (Hybrid OLAP), or MIXED.

In the SAS/Warehouse Administrator Explorer, an OLAP Group can be added only to a Subject. Display 4.7 on page 48 shows an OLAP Group (HOLAP Group 2) that contains an OLAP MDDB (OG2 Sum10) and a OLAP Table (OG2 Sum 12).

Display 4.7 OLAP Group in the Explorer



An *OLAP MDDB* is 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.

An *OLAP Table* is 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.

For details, see Chapter 10, "Maintaining Detail Logical Tables and Detail Tables," on page 155.

Which Summary Data Store Do I Need?

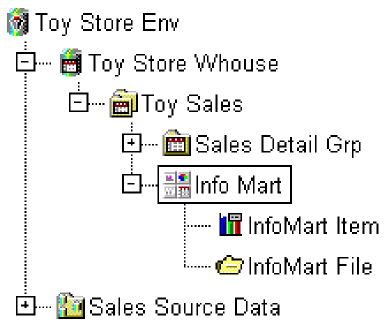
A Data Table can be used as a simple summary data store if you write your own summarization code. Otherwise, the OLAP objects might be more useful. More specifically, use an OLAP Group of type

- ROLAP to support online analytical processing performed on a relational database, such as a SAS table or an ORACLE table
- MOLAP to support online analytical processing performed on a SAS multidimensional database (MDDB)
- HOLAP to support online analytical processing performed on a integrated data store consisting of one or more SAS MDDBs and one or more relational databases
- MIXED to support online analytical processing performed on individual (unrelated) OLAP Tables and OLAP MDDBs.

Information Marts (optional)

An InfoMart is a simple grouping element for InfoMart Items and InfoMart Files, as shown in Display 4.8 on page 49.

Display 4.8 Information Mart in the Explorer



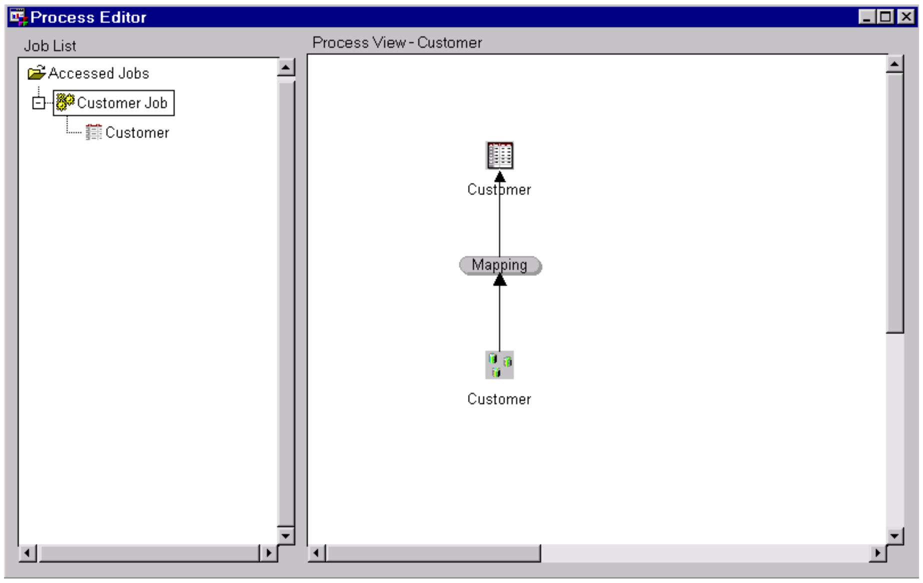
Unlike most objects in SAS/Warehouse Administrator, InfoMart Items and Files are used to display information rather than store it. For example, an InfoMart Item might be used to display a chart that summarizes sales information from a warehouse data store. An InfoMart File might be used to open a spreadsheet that contains information useful to the person managing a given Warehouse Environment. In the SAS/Warehouse Administrator Explorer, an InfoMart can be added only to a Subject, a Data Group, or an ODD Group.

For details, see Chapter 12, "Maintaining Information Marts," on page 239.

Jobs

After using the Explorer to define the properties of a given data store, you will use the Process Editor to define its Job. A *Job* is a metadata record that specifies the processes that create one or more data stores. In Display 4.9 on page 50, a Job that creates the Customer Detail Table is represented by the icon with the rectangle around it in the left panel of the Process Editor.

Display 4.9 Job and Process Flow in the Process Editor



To create the Customer table, you would click the Customer Job with the left mouse button and select **`Run`** from the pop-up menu. The Load Generation/Execution Properties window would display. From the Load Generation/Execution Properties window, you could execute the code associated with the Customer Job.

If you want SAS/Warehouse Administrator to generate the code for a Job,

□ the Job's **`SAS/Warehouse Administrator Generated`** attribute must be set (it is set by default)

□ you must create a Process Flow for the Job in the right panel of the Process Editor.

If the Job's **`SAS/Warehouse Administrator Generated`** attribute is set, SAS/Warehouse Administrator generates code for each process and data store in the Process Flow, from the bottom to the top.

If you want to specify your own code for a Job,

□ the Job's **`User Written`** attribute must be set

□ the Job must specify the location of the user-written source code to be executed for this Job.

If the Job's **`User Written`** attribute is set, SAS/Warehouse Administrator retrieves code that creates the data store(s) for that Job. If you will supply the source code for a Job, no Process Flow is required, although you might want to create one for documentation purposes.

A Job can include scheduling metadata that enables the Process Flow or user-supplied program to be executed in batch mode at a specified date and time. The steps for creating Jobs are described in Chapter 13, "Maintaining Jobs," on page 251.

Job Inputs and Outputs

Keep the following in mind as you plan Jobs for which SAS/Warehouse Administrator will generate code:

- The code generated for a Job only creates the data stores that are specified as *output tables* for the Job. It does not create every data store in the Process Flow in the right panel of the Process Editor.
- The output tables for a Job are listed under the Job's icon in the left panel of the Process Editor. Other data stores in the Process Flow are *inputs* to the Job.

  For example, in Display 4.10 on page 51, the code generated for the All Sales Detail Job will only create Sales Detail Grp, Customer, Drop, and so on. It will not create the ODDs in the Process Flow (icons at the bottom of the Process Flows in the right panel of the Process Editor). The ODDs are inputs to the All Sales Detail Job. They must be created in one or more separate Jobs. After the ODD Jobs have been executed, the ODDs can supply information to the output tables in the All Sales Detail Job.

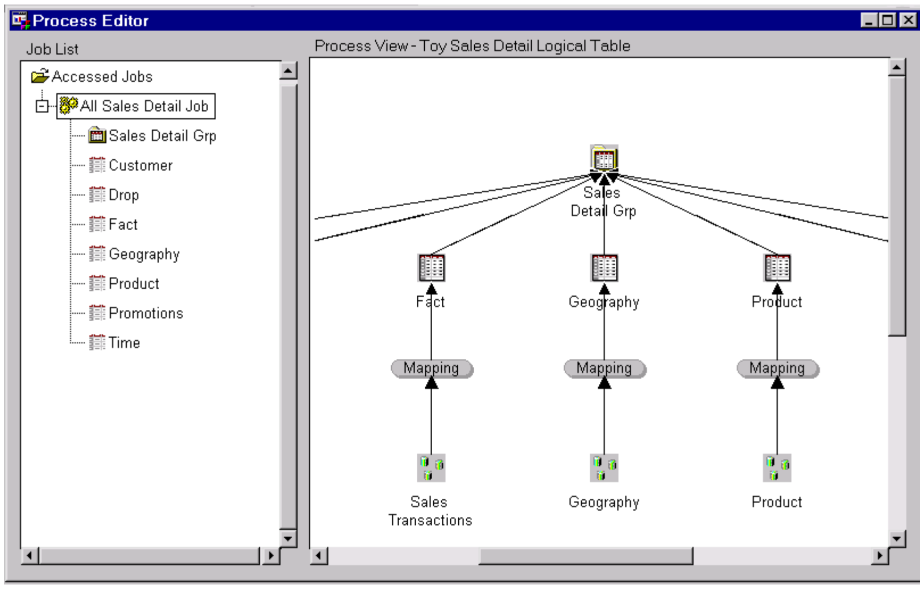
Creating Multiple Output Tables with One Job

You do not have to create a Job for each data store in your project. You can create a single Job that creates multiple data stores.

For example, Display 4.6 on page 47 shows a Detail Logical table (Sales Detail Grp) with Detail Tables (Customer, Drop, and so on). You could define a series of individual Jobs, such as the Customer Job shown in Display 4.9 on page 50, which would create each of the Detail Tables under Sales Detail Grp.

Alternatively, you could create a single Job that would create Sales Detail Grp and all of its Detail Tables, as shown in Display 4.10 on page 51.

Display 4.10 Job with Multiple Output Tables in the Process Editor



To create all of the output tables listed under the All Sales Detail Job (item with the rectangle around it in the left panel of the Process Editor), you would click the Job with the left mouse button and select **`Run`** from the pop-up menu. The Load

Generation/Execution Properties window would display. From the Load Generation/Execution Properties window, you could execute the code associated with the All Sales Detail Job.

For more details, see “Example: Defining a Job with Multiple Output Tables and Input Sources in a Process Flow” on page 265.

Restrictions on Jobs with Multiple Output Tables

SAS/Warehouse Administrator does not allow you to add an output table that is invalid for a particular Job. All output tables must be in the same metadata repository as the Job.

For example, you can create a Job whose output tables are all in the same Data Warehouse, and you can create another Job whose output tables are all ODDs in the same Warehouse Environment. But, you cannot create a Job in which both a warehouse data store and its ODD are specified as outputs.

For example, if you had to create a set of Jobs to manage the sources and targets shown in Display 4.10 on page 51, you could

1. Create and run one Job called All Sales ODDs that creates and loads all of the toy sales ODDs in the Toy Store Environment.
2. Create and run another Job such as the All Sales Detail Job shown in Display 4.10 on page 51.

For details about the impact of metadata repositories on some user operations, see “Overview: Metadata Repositories” on page 313.

Processes

Most of the work in defining a Job is in defining its processes. A process is a routine that creates a warehouse data store or that extracts data, transforms data, or loads data into a data store. In the Process Editor, you define metadata records that are used to generate or retrieve the source code for processes. Mappings, User Exits, Data Transfers, Record Selectors, and Load Steps are all metadata records that generate or retrieve processes.

Each process that you define in the Process Editor generates or retrieves code. SAS/Warehouse Administrator can generate source code for any process except a User Exit, a Detail Logical Table Load Step, or an ODD Load Step. However, you can specify a user-written routine for any process.

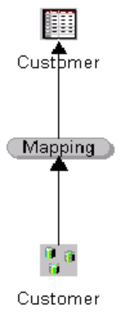
The sections that follow give an overview of SAS/Warehouse Administrator processes.

Mappings

A Mapping process is automatically added when you specify an input for most data stores in the Process Editor. A 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.

For example, in Display 4.11 on page 53, columns from a source table named *Customer* are mapped to columns in a target table named *Customer*.

Display 4.11 Process Flow With Mapping in the Process Editor

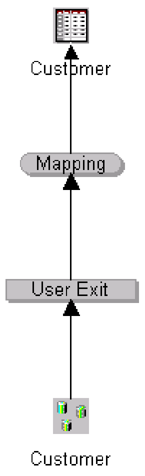


Common mappings include one-to-one (one data source to a target table), joins (one or more data sources merged by one or more common columns), and unions (two or more data sources appended to a target table). For details, see the two Mapping examples in Chapter 14, “Maintaining Processes,” on page 281.

User Exits

A *User Exit process* is a metadata record used to retrieve a user-written routine. The routine must be stored 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. Display 4.12 on page 53 shows a User Exit that could run a validation routine on customer income levels before loading them into a warehouse table.

Display 4.12 Process Flow With User Exit in the Process Editor



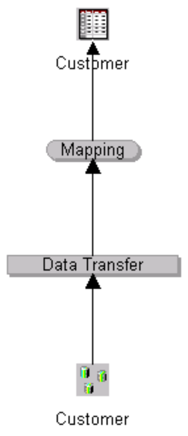
For details, see “Example: Defining User Exit Process Properties” on page 290.

Data Transfers

A Data Transfer process is a metadata record 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.

For example, in Display 4.13 on page 54, suppose that the Customer source table was on Host 1, and the Customer target table was on Host 2.

Display 4.13 Process Flow With Data Transfer in the Process Editor



The Process Flow in Display 4.13 on page 54 illustrates how a Data Transfer could be used to copy the Customer source data on Host 1 to an intermediate work table on a Host 2. The Mapping in the Process Flow would then map columns from the intermediate work table to the target table.

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. For details, see "Example: Defining Data Transfer Process Properties to Move Data from Remote Host to Local Host" on page 294.

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.

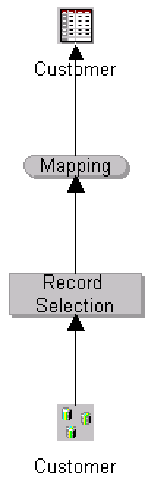
Note: Data Transfers always execute on the remote host (a host other than the host where SAS/Warehouse Administrator is installed). △

Record Selectors

A *Record Selector process* is a metadata record used to generate or retrieve a routine that subsets data prior to loading it to a specified table. In the current release, a Record Selector can be used only to subset the source data specified in an ODD or in a Data File (which is an input to an ODD).

For example, Display 4.14 on page 55 illustrates a Record Selector process that subsets information from the Customer source.

Display 4.14 Process Flow With Record Selector in the Process Editor



For details, see “Example: Defining Record Selector Process Properties” on page 298.

Load Processes

A *Load process* (also called a Load Step) is a metadata record 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 actually creates and loads the data store.

Unlike data preparation processes (Mappings, User Exits, Data Transfers, and Record Selectors), Load processes do not have their own icons in a Process Flow. They are associated with the icons of the tables they create. For example, in Display 4.14 on page 55, the Load processes for the Customer source table and the Customer target table are not shown in the Process Flow.

To view the Load process metadata for a given data store:

1 Display the Process Flow for the data store.
2 In the Process Flow, click the right mouse button on the data store whose Load process you would like to inspect. Select `Edit Load Step` from the pop-up menu.

   A window will display the Load process metadata for the selected data store.

For details, see the Load process examples in Chapter 14, “Maintaining Processes,” on page 281.

Valid Inputs and Outputs for Data Stores

If you want SAS/Warehouse Administrator to generate the code for a Job, you must create a Process Flow such as the one shown in Display 4.14 on page 55. 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 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.

To create a Process Flow, click an object in the Process View with your right mouse button, then select a valid object to add. The software will not let you add an invalid object. The table in this section lists the valid inputs and outputs for data stores and processes in the Process Editor.

The column headings in the table are as follows:

Data Store	is the data store type.
Inputs	are valid inputs for the data store in the Process Editor. *Note:* Some data stores can use the old Summary Tables or Summary MDDBs as input. However, no current data store can specify a Summary Table or a Summary MDDB as an output target. (You cannot create these old summary objects.) △
Outputs	are the valid outputs for the data store in the Process Editor.
Processes	are the process types that can be defined for the data store.

Table 4.1 Valid Inputs and Outputs for Data Stores

Data Store	**Inputs**	**Outputs**	**Processes**
Data File	a SAS data source	ODD, Data File	User Exit, Data Transfer, Record Selector
External File	a data source other than SAS	ODD, Data File	User Exit, Data Transfer
ODD	Data File, External File	Detail Table, Data Table, OLAP Group, OLAP Table, OLAP MDDB, Information Mart Item, Information Mart File	User Exit, Data Transfer, Record Selector, Load
Data Table	Detail Table, Data Table, Detail Logical Table, Summary Table, Summary MDDB, OLAP Table, OLAP MDDB, OLAP Group, ODD	Detail Table, Data Table, OLAP Group, OLAP Table, OLAP MDDB, Information Mart Item, Information Mart File	Mapping, User Exit, Data Transfer, Load
Detail Table	Detail Table, Data Table, Detail Logical Table, Summary Table, Summary MDDB, OLAP Table, OLAP MDDB, OLAP Groups, ODD	Detail Table, Detail Logical Table, Data Table, Summary Table, Summary MDDB, OLAP Group, OLAP Table, OLAP MDDB, Information Mart Item, Information Mart File	Mapping, User Exit, Data Transfer, Load
Detail Logical Table	Detail Table	Detail Table, Data Table, Summary Table, Summary MDDB, OLAP Group, OLAP Table, OLAP MDDB, Information Mart Item, Information Mart File	User Exit, Data Transfer, Load

Data Store	Inputs	Outputs	Processes
OLAP Group	Detail Table, Data Table, Detail Logical Table, Summary Table, Summary MDDB, OLAP Table, OLAP MDDB, OLAP Groups, ODD	Detail Table, Data Table, OLAP Group, OLAP Table, OLAP MDDB, Information Mart Item, Information Mart File	Mapping, Load
OLAP Table	Detail Table, Data Table, Detail Logical Table, Summary Table, Summary MDDB, OLAP Table, OLAP MDDB, OLAP Groups, ODD	Detail Table, Data Table, OLAP Group, OLAP Table, OLAP MDDB, Information Mart Item, Information Mart File	Mapping, User Exit, Data Transfer, Load
OLAP MDDB	Detail Table, Data Table, Detail Logical Table, Summary Table, Summary MDDB, OLAP Table, OLAP MDDB, OLAP Groups, ODD	Detail Table, Data Table, OLAP Group, OLAP Table, OLAP MDDB, Information Mart Item, Information Mart File	Mapping, User Exit, Data Transfer, Load
Information Mart File (within a Data Warehouse)	Detail Table, Data Table, Detail Logical Table, Summary Table, Summary MDDB, OLAP Table, OLAP MDDB, OLAP Groups, ODD	NA	Load
Information Mart File (within an ODD)	Data File, External File, ODD	NA	Load
Information Mart Item (within a Data Warehouse)	Detail Table, Data Table, Detail Logical Table, Summary Table, Summary MDDB, OLAP Table, OLAP MDDB, OLAP Groups, ODD	NA	Load
Information Mart (within an ODD)	Data File, External File, ODD	NA	Load

What's Next

After you have identified the groups, data stores, and processes that you need to implement a data collection, you are ready to implement it in SAS/Warehouse Administrator.

PART 3

Implementation

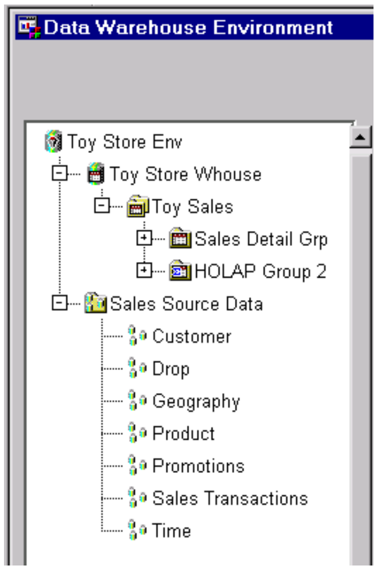
CHAPTER
5

Maintaining Environments

Overview

After you have created a project plan for a data warehouse or a data mart, you are ready to begin work in SAS/Warehouse Administrator. For each data warehousing project, you will create a hierarchy of groups and data stores in the SAS/Warehouse Administrator Explorer, such as the hierarchy that is shown in Display 5.1 on page 62.

Display 5.1 Warehouse Environment in the Explorer



The first object that you create will be a Data Warehouse Environment. A Data Warehouse Environment is 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 Operational Data Definition Groups (ODD Groups).

In a broad sense, creating a Warehouse Environment includes defining the global metadata and the data sources for the Environment. These topics are covered in other chapters. This chapter focuses on the Warehouse Environment object that defines the _MASTER metadata repository.

To create a Warehouse Environment:

1. Create a directory structure for the Warehouse Environment.
2. Define the properties of the Warehouse Environment.
3. Create a start method for the Environment, if one is required.

Note: The basic steps for maintaining Warehouse Environments are described in the online Help. To display the relevant online help, in the SAS Help contents, select **`Help on SAS Software Products`**, then select **`Using SAS/Warehouse Administrator Software`**. Select **`Setting Up Your Warehouse Environment`**, then **`Maintaining Warehouse Environments`**. In addition, you can display Help for most SAS/Warehouse Administrator windows by selecting Help on the window. △

Working with Existing Environments

Adding the Example Environment

When SAS/Warehouse Administrator is installed, an example Warehouse Environment is also installed. For details about adding the example Environment to the SAS/Warehouse Administrator desktop, see Appendix 2, "Adding the Example Environment," on page 333.

Metadata Conversion Wizard

The Metadata Conversion Wizard converts Release 1.x metadata to Release 2.0 metadata. If you have any Data Warehouse Environments that were created with a previous release of SAS/Warehouse Administrator, you must convert the metadata in these old Environments to Release 2.0 format in order to use these Environments in the current release.

To invoke the Metadata Conversion Wizard, run SAS/Warehouse Administrator Release 2.0, add a new Environment to the desktop, and in the **Path** field for the new Environment, specify the path to a Release 1.x Environment's metadata repository that has not been converted to Release 2.0 format. The Metadata Conversion Wizard displays.

For more details, see the wizard's online documentation. First-time users of the Metadata Conversion Wizard might want to read Appendix 1, "Converting Metadata for Environments and Warehouses," on page 323.

Opening a Warehouse Environment in the Explorer

Here is one way to open a Warehouse Environment in the SAS/Warehouse Administrator Explorer:

1. Run SAS on a machine where SAS/Warehouse Administrator has been installed.
2. Type **dw** on the SAS command line, and press RETURN. SAS/Warehouse Administrator will open the default desktop.
3. On the SAS/Warehouse Administrator desktop, position the cursor on the Environment icon, click your right mouse button, and select **Edit** from the pop-up menu. (Under Microsoft Windows and OS/2 operating environments, you can double-click the Environment icon to open the Environment.)

The Environment will be opened in the SAS/Warehouse Administrator Explorer. For details about the SAS/Warehouse Administrator Explorer window, click Help in that window.

Opening an Environment with a Relative Pathname

If the pathname in an Environment's **Path** field is a fully qualified path on the local host, you can start SAS in any convenient directory.

If the pathname in an Environment's **Path** field is a relative path on the current host, start SAS in a directory that will resolve the pathname to the metadata library _MASTER. Alternatively, you can change to the appropriate parent directory by entering a command such as the following on the SAS command line:

```
x 'cd drive_name:\dir_name'
```

If you try to open a Warehouse Environment and you get an error in the SAS log, such as

```
ERROR: Invalid physical name for library _MASTER
```

this might indicate that SAS cannot resolve a relative pathname to the Warehouse Environment that you are trying to open. As a result, the groups and data stores in that Environment will not be visible in the SAS/Warehouse Administrator Explorer. If you get this error, ask the person who created the Environment to identify the appropriate parent directory that will resolve the Environment's relative pathname.

If the pathname in an Environment's **`Path`** field is a relative path on a remote host and the Environment is under the control of a SAS/SHARE server, see "SAS/SHARE Server Preparation" on page 68.

Metadata Copy Wizard

The Metadata Copy Wizard copies a Warehouse Environment to a new location that you specify. You can use this feature to model a new Environment after an old one rather than creating a completely new Environment. To invoke the Metadata Copy Wizard, run SAS/Warehouse Administrator Release 2.0. On the SAS/Warehouse Administrator desktop, click an Environment's icon with the right mouse button and select **`Copy`**. The Metadata Copy Wizard displays. Read the wizard's online documentation for details.

Preparing to Create Local or Remote Warehouse Environments

Before you create a Warehouse Environment, you must do some preparation:

Metadata host configuration	Verify that you have the appropriate hardware and software to implement your Environments, as described in "Metadata Host Configuration" on page 23.
Physical paths	The physical path for the metadata library _MASTER must exist. Each Warehouse Environment and each Data Warehouse within an Environment should have a unique pathname. You might also want to create a directory structure for the Environment, as described in "Creating a Directory Structure for a New Environment" on page 64.

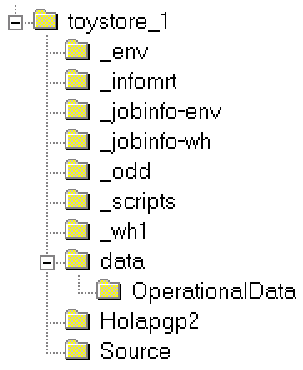
Creating a Directory Structure for a New Environment

Before you enter the metadata for a new Warehouse Environment, you might want to create a top-level directory and a few typical subdirectories for the Environment. At a minimum, you will need subdirectories for a Warehouse Environment and its Data Warehouses.

Note: Each Warehouse Environment—and each Data Warehouse within an Environment—should have a unique pathname. △

Display 5.2 on page 65 illustrates one possible directory structure for the Toy Store Environment that is used as the main example in this document.

Display 5.2 Example Directory Structure



```
toystore_1
    _env
    _infomrt
    _jobinfo-env
    _jobinfo-wh
    _odd
    _scripts
    _wh1
    data
        OperationalData
    Holapgp2
    Source
```

The directory structure in Display 5.2 on page 65 is not required. It is simply an example of a valid structure. It is a convenient way to keep metadata, source libraries, scripts, and other elements that are associated with a given Environment together. The directories in the example are as follows:

toystore_1	is a top-level directory that includes some local subdirectories that are associated with the Toy Store Environment. SAS/Warehouse Administrator does not require a top-level directory such as toystore_1, but the combination of a top-level directory structure and the use of relative pathnames in SAS/Warehouse Administrator will make it easier to copy or move a Warehouse Environment—as in a test-to-production scenario.
_env	is the directory for the Toy Store Environment and its metadata repository (reserved libref _MASTER). This library contains host definitions and other metadata shared by all Warehouses and ODD Groups in a given Environment. You should have a unique directory for each Environment in your project.
_infomrt	is a directory for a SAS library that contains the example Information Mart Item and File described in Chapter 12, "Maintaining Information Marts," on page 239. You might need Information Mart Items or Files for your project.
_jobinfo-env	is a directory for a Jobs Information library. You do not need such a directory unless you are using SAS/Warehouse Administrator to schedule and track Jobs. A Warehouse Environment and its Data Warehouses can share one Jobs Information library, or they could have separate Jobs Information libraries. For details, see Chapter 15, "Scheduling Jobs," on page 305.
_jobinfo-wh	is a directory for another Jobs Information library.
_odd	is a directory where any local tables or views that are associated with Operational Data Definitions (ODDs) can be stored. You could use such a directory if your ODDs create views or output tables.
_scripts	is a directory where any SAS/CONNECT scripts that are unique to this Environment could be stored. The metadata for a remote host definition includes a reference to a SAS/CONNECT script used to establish a connection with that host. You could use such a directory if any warehouse resources are stored on remote hosts.

_wh1	is the directory for the Toy Store Data Warehouse and its metadata repository (reserved libref _DWMD). The _DWMD library is the metadata repository for most of the groups and data stores in a data warehouse or a data mart within the current Environment. You should have a unique directory for each Data Warehouse in your project.
data	is a directory for a SAS library that contains detail data that is local to the SAS/Warehouse Administrator host. Neither data sources nor data stores must be local to the SAS/Warehouse Administrator host.
Holapgrp2	is a directory for a SAS library that contains summary data that is local to the SAS/Warehouse Administrator host.
Source	is a directory where a SOURCE catalog containing user-written routines that are unique to this Environment could be stored. These routines are used to supplement or replace the code that is generated by SAS/Warehouse Administrator. For example, user-written Load Step routines for ODDs could be stored in the SOURCE catalog, because SAS/Warehouse Administrator does not generate Load Steps for ODDs.

Specifying Physical Pathnames in SAS/Warehouse Administrator

The metadata for SAS libraries and other objects in SAS/Warehouse Administrator include a **Path** field where you specify a pathname for the objects. Remember the following things when specifying a physical path for an object in SAS/Warehouse Administrator:

- If the path exists and is local to SAS/Warehouse Administrator, you can use the right arrow button to select the path.
- If the path exists and is remote to SAS/Warehouse Administrator, enter the path as it exists on the remote host. (Do not include the remote host name in the path.)
- If the path does not exist, you must use operating system commands to create the physical path for the object before you can specify its metadata in SAS/Warehouse Administrator.

Pathname Portability

In general, enter physical pathnames that will be reasonably portable, given the operating systems where these objects will be stored. This will allow you to copy or move these objects to different hosts and minimize any updates to the physical pathnames in the metadata.

The least portable pathname might be one that includes a drive specification, such as d:\Dw_projects\Project-1_env.

A UNC (Universal Naming Convention) pathname such as \\pchost1\projects\Dw_projects\Project-1_env is somewhat more portable, as long as the UNC format is appropriate for the host's operating system.

The most portable pathname might be a relative pathname such as .\Project-1_env. However, you must be sure that the current directory for the SAS session will resolve any relative pathnames for SAS/Warehouse Administrator objects. For details about the possible impact of relative pathnames, see "Opening an Environment with a Relative Pathname" on page 63.

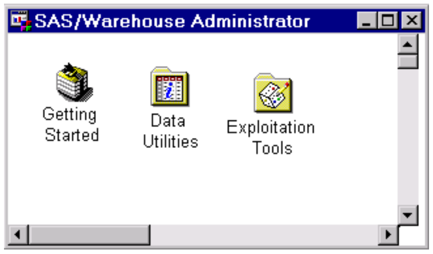
Example: Creating a Local Warehouse Environment

This example summarizes how to create a local Warehouse Environment—an Environment whose metadata repository is stored on the SAS/Warehouse Administrator host. The following explanations describe the metadata and methods used to achieve the desired results.

Define Warehouse Environment Properties

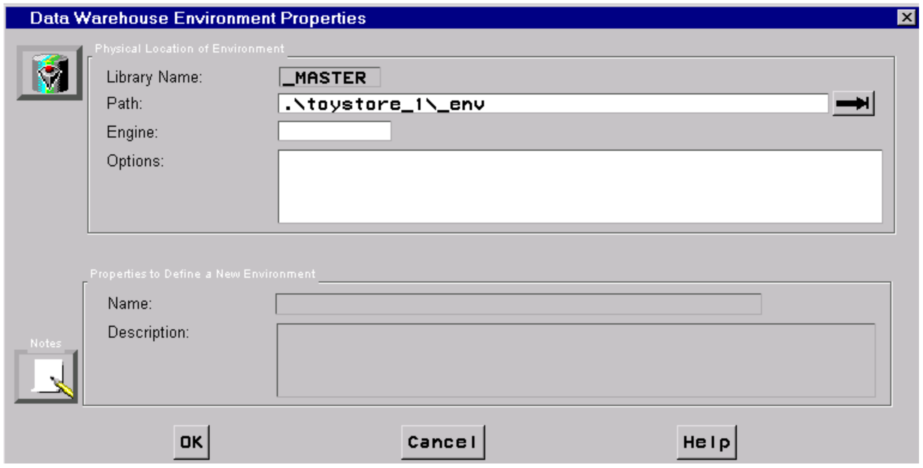
If you have not done so already, display the SAS/Warehouse Administrator desktop. Run SAS on a machine where SAS/Warehouse Administrator has been installed. Type **dw** on the SAS command line, and press RETURN.

The following display illustrates a SAS/Warehouse Administrator desktop that has no Warehouse Environments defined:



Position the cursor on an empty area on the desktop, click the right mouse button, select **Add Item**, and then **Data Warehouse Environment**. A properties window for the Environment displays as seen in the following display. Enter the appropriate information as follows:

Path	specify the physical path where the Environment's metadata repository (_MASTER) will be stored.


Data Warehouse Environment Properties
Physical Location of Environment
Properties to Define a New Environment
Notes
OK
Cancel
Help

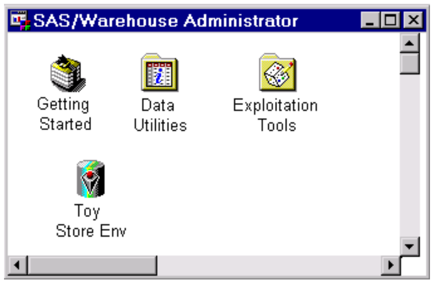
Library Name: _MASTER
Path: .\toystore_1_env
Engine:
Options:
Name:
Description:

The **Path** field in this example contains a relative pathname that is appropriate for a SAS library on a PC (**.\toystore_1_env**). For details about specifying pathnames, see "Specifying Physical Pathnames in SAS/Warehouse Administrator" on page 66.

The **Engine** and **Options** fields can be left blank. After specifying a path, click OK.

If you are creating a new Environment, you will be prompted to enter a **Name** and a **Description**. After specifying a name and description, click OK. The new Environment is added to the SAS/Warehouse Administrator desktop.

The following display illustrates a SAS/Warehouse Administrator desktop that has the Toy Store Environment defined:



You are now finished defining the properties for this Environment. To ensure that you can start SAS using a configuration file and an autoexec file that are appropriate for this Environment, see "Creating a Start Method for an Environment" on page 71.

Example: Creating a Remote Warehouse Environment

This example summarizes how to create a remote Warehouse Environment—an Environment whose metadata repository is not stored on the SAS/Warehouse Administrator host.

A remote Environment might be appropriate if you require concurrent read/write access to Environments by multiple SAS/Warehouse Administrator hosts. In that case, you must create a remote Environment and put its metadata repository under the control of a SAS/SHARE server that is remote to the SAS/Warehouse Administrator hosts. For an example of such a configuration, see "Remote Metadata: PC Client to Windows NT Server" on page 25.

The following sections describe the metadata and methods that are used to achieve the desired results. The appropriate preparation is assumed to have taken place, as described in "Preparing to Create Local or Remote Warehouse Environments" on page 64. Some additional preparation for remote Environments is described in the next section.

SAS/SHARE Server Preparation

On the SAS/SHARE server host,

- create a directory structure for the Environment
- assign a SAS libref to the directory that corresponds to the _MASTER library for the Environment; do this in such a way that the library is under the control of the SAS/SHARE server.

For example, suppose that you created the following directory structure on the SAS/SHARE server host,

```
.\Project-2\_env
.\Project-2\_wh1
```

where .\Project-2_env is the directory that corresponds to the _MASTER metadata repository for the Environment that you will create. You would then assign a libref to that directory in such a way that the library is under the control of the SAS/SHARE server.

Note: Because this is a SAS/SHARE libref and not a SAS/Warehouse Administrator libref, you are not restricted to _MASTER as the libref. For example, to the SAS/SHARE server, the .\Project-2_env directory could have a SAS/SHARE libref of ENV1. (The SLIBREF option would then have to be used in the Warehouse Environment's metadata, as described in "Define Warehouse Environment Properties" on page 69.) △

Note: If you use a relative pathname such as .\Project-2_env for the Environment library, be sure that the SAS/SHARE server can resolve the pathname. △

SAS/Warehouse Administrator Client Preparation

Verify that your local SAS session can access the remote library that will ultimately become the metadata repository for the Environment that you are creating. One way to do that is to submit a LIBNAME statement for that library. Here is an example of such a statement,

```
libname _MASTER server=host2.dwshare slibref=ENV1;
```

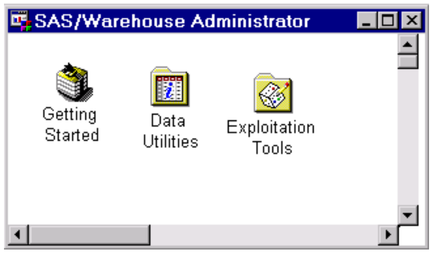
where **`_MASTER`** is the local libref for the Warehouse Environment's metadata repository, **`host2`** is the name of the remote host, **`dwshare`** is the name of the SAS/SHARE server, and **`slibref=ENV1`** is the remote server's libref for the Warehouse Environment's metadata repository.

If you can successfully execute such a statement, then SAS/Warehouse Administrator can successfully generate code for the remote Warehouse Environment.

Define Warehouse Environment Properties

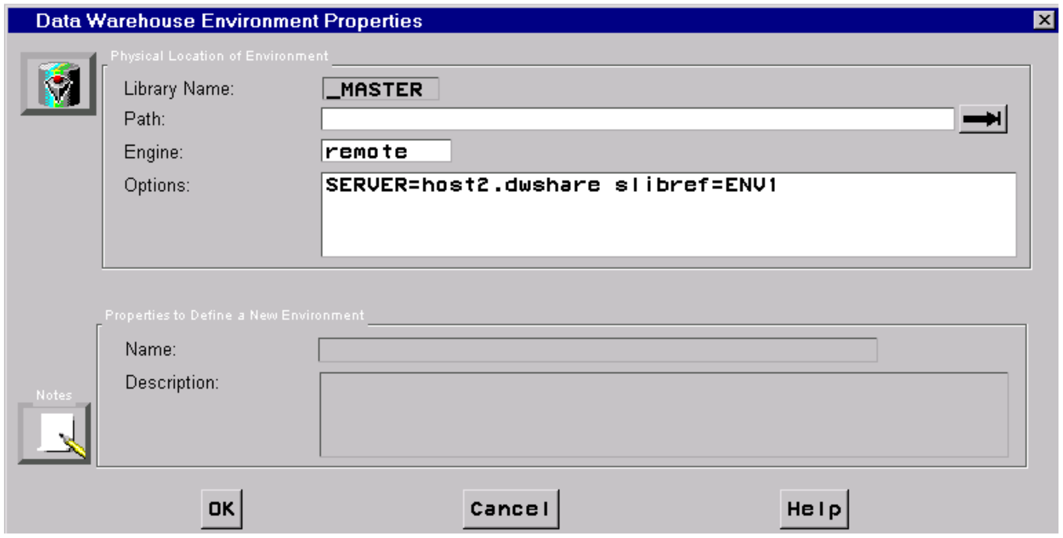
If you have not done so already, display the SAS/Warehouse Administrator desktop. Run SAS on a machine where SAS/Warehouse Administrator has been installed. Type **`dw`** on the SAS command line, and press RETURN.

The next display illustrates a SAS/Warehouse Administrator desktop that does not have any Warehouse Environments defined:



Position the cursor on an empty area on the desktop, click the right mouse button, select **`Add Item`**, then **`Data Warehouse Environment`**. A properties window for the Environment displays for you to enter the appropriate information, which is described as follows:

`Engine`	specify a SAS engine appropriate for the current Environment.
`Options`	specify any LIBNAME options appropriate for the current Environment.


Data Warehouse Environment Properties
Physical Location of Environment
Properties to Define a New Environment
Notes
OK
Cancel
Help

Library Name: _MASTER
Path:
Engine: remote
Options: SERVER=host2.dwshare slibref=ENV1
Name:
Description:

In the current example, you do not enter a physical path in the **Path** field because the _MASTER library will be accessed through the SAS/SHARE server, which uses a libref to access this library.

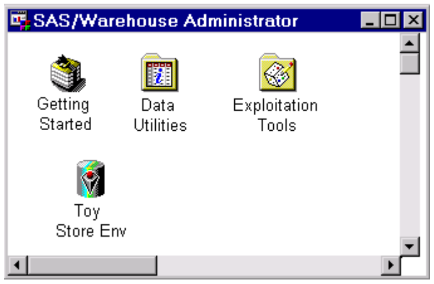
The **Engine** field specifies that the remote SAS engine will be used to access the _MASTER library for this Environment.

The **Options** field specifies the SERVER= option required to access the _MASTER library for this Environment, where **host2** is the name of the remote host, **dwshare** is the name of the SAS/SHARE server, and **slibref=ENV1** is the remote server's libref for the Warehouse Environment's metadata repository.

After specifying the appropriate values, click $\boxed{\text{OK}}$.

If you are creating a new Environment, you will be prompted to enter a **Name** and a **Description**. After specifying a name and description, click $\boxed{\text{OK}}$. The new Environment is added to the SAS/Warehouse Administrator desktop.

The next display illustrates a SAS/Warehouse Administrator desktop that has the Toy Store Environment defined:



You are finished defining the properties for this Environment. To ensure that you can start SAS with a configuration file and an autoexec file appropriate for this Environment, see "Creating a Start Method for an Environment" on page 71.

Creating a Start Method for an Environment

SAS/Warehouse Administrator runs in a SAS session. The attributes of the underlying SAS session should be appropriate for the tasks you want to perform in a particular Warehouse Environment.

Note: When you create a given Warehouse Environment, you should provide the means to start SAS with a configuration file and an autoexec file appropriate for the Environment. △

It is recommended that you create an appropriate start method for any Environment that you create, so that anyone who uses your Environment will have the correct session attributes.

Creating a Shortcut on a PC Desktop

Suppose that you have installed SAS/Warehouse Administrator on a Microsoft Windows NT host, and you have created a Warehouse Environment on that host. The start method for that Environment might be a desktop shortcut, which could be created as follows:

1 Click the right mouse button on the desktop and select **`New`**, and then **`Shortcut`**.

2 In the Create Shortcut window, click Browse to select the appropriate sas.exe file.

   When you have selected the file, click Next.

3 In the next window, enter a name for the shortcut and click Finish.

   The shortcut is created on the desktop. The next task is to specify a SAS configuration file and a SAS autoexec file appropriate for the tasks you want to perform in SAS/Warehouse Administrator.

4 Click the shortcut icon with the right mouse button and select **`Properties`**.

   The Properties window for the shortcut displays.

5 In the Properties window for the shortcut, click the Shortcut tab.

   The Shortcut tab displays. The **`Target`** field specifies the sas.exe file that you selected in Step 2. For example,

```
D:\sas8\sas.exe
```

6 On the Shortcut tab, specify a SAS configuration file and a SAS autoexec file appropriate for the tasks you want to perform in SAS/Warehouse Administrator in the **`Target`** field. For example,

```
''D:\sas8\sas.exe'' -config ''d:\username\config\sasv8.cfg''
-autoexec ''d:\username\config\autoexec.sas''
```

   You could replace **`username`** with any convenient directory name.

   Note: To handle spaces in pathnames, use double quotation marks to delimit all paths. △

   For details about SAS configuration files and SAS autoexec files appropriate for SAS/Warehouse Administrator, see “SAS Startup Files and SAS/Warehouse Administrator” on page 72.

7 To start SAS in a particular directory, specify that directory in the **`Start in`** field. For example,

```
D:\username\dw\demo
```

   This is important if the Warehouse Environment that you created uses relative pathnames. For details, see “Opening an Environment with a Relative Pathname” on page 63.

8 When you are finished, click OK to save your changes.

The shortcut is ready to use.

SAS Startup Files and SAS/Warehouse Administrator

SAS Configuration File

When you install SAS software and SAS/Warehouse Administrator, the installation process creates a SAS configuration file (such as sasv8.cfg) that includes the appropriate SASHELP catalogs into the concatenated SASHELP library.

Be sure to specify the appropriate SAS configuration file in your SAS start command so that the SAS/Warehouse Administrator SASHELP catalogs will be visible in the underlying SAS session.

You might want to specify SAS options in the configuration file that would be useful in a SAS/Warehouse Administrator session. For details about such options, see “SAS System Options and SAS/Warehouse Administrator” on page 73.

SAS Autoexec File

A SAS autoexec file can include LIBNAME statements, SAS options, and various commands that are appropriate for the tasks you want to perform in SAS/Warehouse Administrator. For example, here is a typical SAS autoexec file for SAS running under Microsoft Windows or Windows NT:

```
/* Typical SAS autoexec for SAS/Warehouse Administrator */

/* Assign getusrpw catalog. Provides access */
/* to GETUSRPW macro required by some */
/* SAS/CONNECT scripts. */
filename getusrpw catalog 'sashelp.dwport';
options sasautos=(getusrpw sasautos);

/* Assign _saswa lib. Provides access */
/* to SAS/Warehouse Administrator add-ins */
/* and other software. */
libname _saswa 'd:\username\dw\sample8';

/* Start SAS/Warehouse Administrator */
dm "dw" continue;
```

GETUSRPW support is often enabled on the host where SAS/Warehouse Administrator is installed. However, the getusrpw catalog is not needed until you are ready to create and test remote host definitions.

Most sites will want to install and use at least some of the add-in software available for SAS/Warehouse Administrator. However, the _saswa library is not needed until you have installed some of these add-ins. In the libname _saswa statement above, you could replace username with any convenient directory name.

You might want to have your SAS autoexec file start SAS/Warehouse Administrator, as shown in the previous example (see the **`dm "dw" continue;`** command).

You might want to add a command to your SAS autoexec file that will set the start directory for the SAS session. Here is an example of such a command:

```
x 'cd c:\username\dw'
```

This is important if the Warehouse Environment in which you will be working uses relative pathnames.

You also might want to specify SAS options in the autoexec file that would be useful in a SAS/Warehouse Administrator session. For details about such options, see "SAS System Options and SAS/Warehouse Administrator" on page 73.

SAS System Options and SAS/Warehouse Administrator

When using SAS/Warehouse Administrator, the attributes of the underlying SAS session must be appropriate for the tasks you want to perform. This topic describes the impact of some SAS options on SAS/Warehouse Administrator.

Note: For full details about a SAS option, see the *SAS Language Reference: Dictionary*. △

VALIDVARNAME= Option

The SAS option VALIDVARNAME= controls the type of SAS column names that can be used and created during a SAS session.

VALIDVARNAME=v6
: enforces Version 6 rules for SAS names, where the maximum length is eight characters; all letters are displayed in uppercase, regardless of how they are entered; and maximum width for character columns is 200 characters.

VALIDVARNAME=v7
: enforces Version 7 rules for SAS names, where the maximum length is 32 characters; letters entered in mixed case are displayed in mixed case; and maximum width for character columns is 32,767 characters. This is the default for SAS/Warehouse Administrator.

VALIDVARNAME=UPCASE
: enforces same rules as VALIDVARNAME=V7, except that all letters are displayed in uppercase, regardless of how they are entered.

VALIDVARNAME=ANY
: is not supported in SAS/Warehouse Administrator.

To specify the VALIDVARNAME= option, include the appropriate statement in a SAS invocation, a SAS autoexec file, or a SAS configuration file as follows:

```
option validvarname=option;
```

Note: For details about the VALIDVARNAME= system option, see the *SAS Language Reference: Dictionary*. △

Using VALIDVARNAME=

By default, SAS/Warehouse Administrator Release 2.0 supports 32–byte, mixed-case column names. When entering metadata for a SAS data store name (table, view, or MDDB), for a column in a SAS data store, or for a SAS catalog or catalog entry, the names can now be up to 32 bytes long, and the letters in the name can be entered in mixed-case.

Note: SAS will not let you specify two names that differ only in case. △

For example, SAS/Warehouse Administrator will not let you specify a column `abc` and a column `ABC` in the same table. Accordingly, when you specify SAS column names in SAS/Warehouse Administrator, do not specify two names that only differ in case.

Note: SAS librefs and filerefs are still limited to a maximum of eight characters. △

In addition to its effect on SAS names, the VALIDVARNAME= option can have other significant impacts in a SAS/Warehouse Administrator session. For example,

- if you have warehouse data stores in DBMS format, the VALIDVARNAME= option determines the rules for mapping DBMS column names to SAS column names, or vice versa. For details, see the VALIDVARNAME= option in the SAS/ACCESS documentation for the DBMS.
- the VALIDVARNAME= option can also have impacts on SAS/CONNECT software and PROC SQL Pass-Through views. For details, see the VALIDVARNAME= option in the documentation for these items.

In general, use the default (VALIDVARNAME=v7) when you are specifying metadata for warehouse data that will reside on a SAS Version 7 or later host, and you would like to take advantage of long, mixed-case column names.

Alternatively, if all of your warehouse data will be stored on SAS Version 6 hosts, you might consider using the VALIDVARNAME=v6 option in your SAS/Warehouse Administrator session, which will not allow you to specify SAS column names that will not work on Version 6 hosts.

What's Next

After adding a Warehouse Environment, you are ready to define host definitions and other global metadata for that Environment.

CHAPTER
6

Maintaining Global Metadata

Overview

After you define a Warehouse Environment, you can define metadata records that can be shared by data stores and other elements within that Environment. For example, if Jane Jones is the administrator for all of the data stores in a given Data Warehouse, you can create a contact record for Jane and save it as part of the global metadata for the parent Environment. You could then include Jane's contact record in the metadata for the data stores in the Warehouse without retyping her information each time.

If you create the main global metadata items first, you can simply select them from a list rather than having to stop and create a host definition in the middle of creating a data store, for example. In each Warehouse Environment, you will define at least some of the following global metadata types:

- SAS library definitions (including SAS/ACCESS LIBNAME definitions and Jobs Information library definitions)
- host definitions
- DBMS connection profiles
- contact records
- scheduling server definitions

Note: The basic steps for maintaining global metadata are described in the online Help. To display the relevant online Help, in the SAS Help contents, select **Help on SAS Software Products**, then select **Using SAS/Warehouse Administrator Software**. Select **Setting Up Your Warehouse Environment**, then **Maintaining Warehouse Environments**, then **Maintaining Global Metadata**. In addition, you can display Help for most SAS/Warehouse Administrator windows by selecting Help on the window. △

Using the Define Items Used Globally Window

To add global metadata records (except for Jobs Information libraries):

1. Open the relevant Warehouse Environment in the SAS/Warehouse Administrator Explorer, as described in "Opening a Warehouse Environment in the Explorer" on page 63.
2. From the menu bar, select **File**, then **Setup**.
3. From the **Define Items Used Globally** window, select the type of global metadata you want to add, then click Add.
4. In the Properties window, enter and save the metadata record.

The sections that follow provide details about each kind of global metadata.

SAS Library Definitions

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. They are used to generate LIBNAME statements for the corresponding libraries. SAS library definitions are included in the metadata records for data stores, processes, and Jobs in the current Environment. They are required in order to access source data and to load warehouse data stores.

Preparing to Create a SAS Library Definition

At a minimum, after you create a Warehouse Environment, and before you add any groups or data stores, it is recommended that you add any SAS library definitions that will be needed for the first data stores, processes, and Jobs that you will add to the current Environment.

The metadata for SAS library definitions includes a **`Path`** field where you specify a pathname for the library. This path must exist, or the LIBNAME statement that is generated from the library definition will not work. You might find it convenient to create the physical paths before you start creating the library definition in SAS/Warehouse Administrator. Then you will know what to enter in the **`Path`** field, and you can test the library definition immediately after creating it.

Keep in mind a SAS library definition does not include a host definition. In most cases, 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.

Example: Creating a Local Library Definition

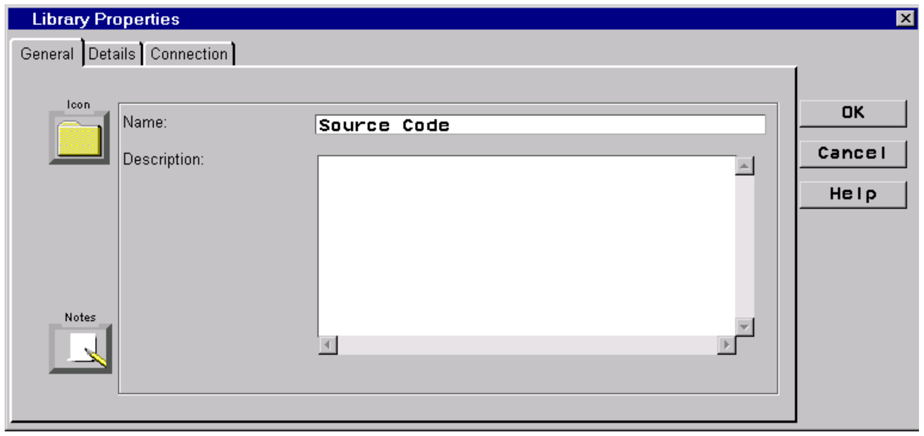
The following example summarizes how to create a metadata record for a SAS library that is stored on the SAS/Warehouse Administrator host. The appropriate Warehouse Environment is assumed to exist.

Define Library Properties

Display the Define Items Used Globally window, as described in “Using the Define Items Used Globally Window” on page 76.

In the Type panel of the Define Items Used Globally window, click the button beside **`SAS Libraries`**. Click Add at the bottom left of the window. A Library Properties window for the library displays for you to enter the appropriate information as follows:

`Name`	specifies a display name for the SAS library you are creating.



Name: Source Code

Description:

The **Name** field contains a display name that is appropriate for a source code library: **Source Code**.

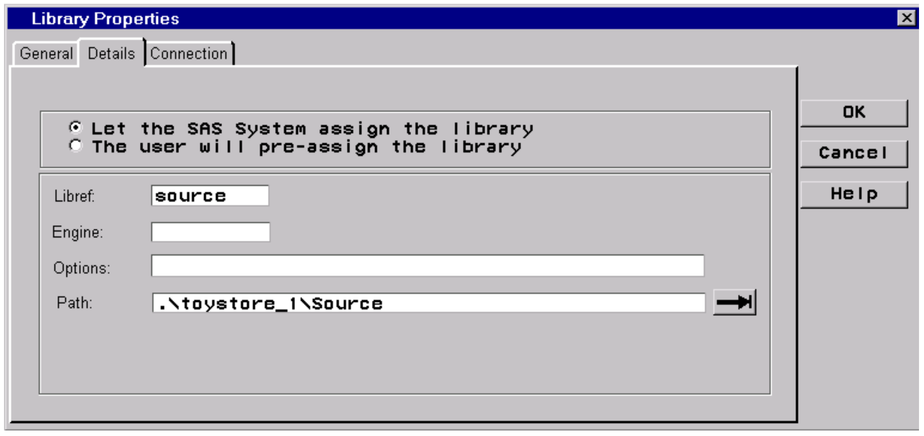
After you have entered a name (and perhaps a description), click the Details tab. The fields on this tab are

Let the SAS System assign the library
: specifies who is responsible for assigning this library—you or SAS/Warehouse Administrator.

  Do not select **Let the SAS System assign the library** if you want to control when this library is assigned.

Libref
: specifies a libref for the library being defined.

Path
: specifies a physical path for the library being defined. For details about specifying pathnames, see "Specifying Physical Pathnames in SAS/Warehouse Administrator" on page 66.



[✓ Let the SAS System assign the library] The user will pre-assign the library

Libref: source

Engine:

Options:

Path: .\toystore_1\Source

The **Let the SAS System assign the library** option is selected in the Details tab above.

The **Libref** field above contains a libref appropriate for a source code library: **source**.

For a local library, you can often leave the **Engine** and **Options** fields blank.

The **Path** field above contains a relative pathname that is appropriate for a SAS library on a PC: **.\toystore_1\Source**.

After specifying values in these fields, click OK. The new library definition is added to the list of libraries in the Define Items Used Globally window.

Example: Creating a Remote Library Definition

This example summarizes how to create a metadata record for a SAS library that is not stored on the SAS/Warehouse Administrator host. The appropriate preparation is assumed to have taken place, as described in "Preparing to Create a SAS Library Definition" on page 77.

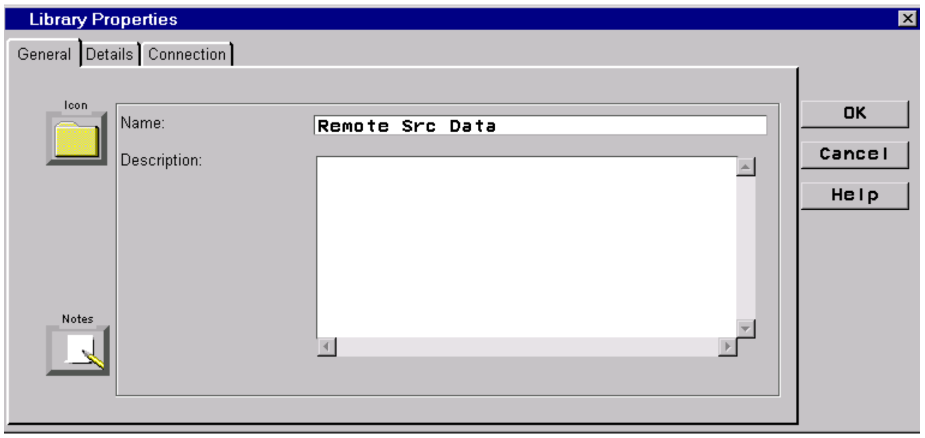
Keep in mind that the physical path that you specify in this library definition will not resolve until you combine the library definition with a host definition for the computer where the library resides. In the metadata for data stores and other objects, you will specify both the library definition and the host definition for the computer where the library resides.

Define Library Properties

Display the Define Items Used Globally window, as described in "Using the Define Items Used Globally Window" on page 76.

In the Type panel of the Define Items Used Globally window, click the button beside **SAS Libraries**. Click Add at the bottom left of the window. A Library Properties window for the library displays for you to enter the appropriate information as follows:

Name	enter a display name for the SAS library you are creating.

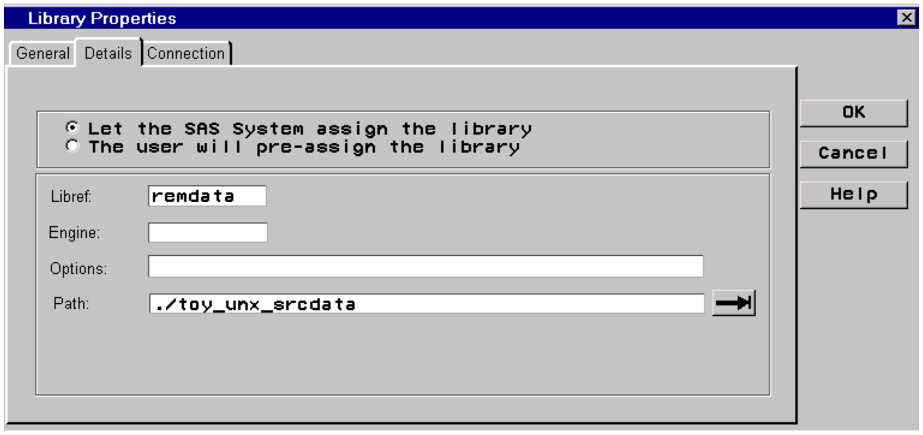


Name: Remote Src Data
Description:

The **Name** field contains a display name that is appropriate for a remote source data library: **Remote Src Data**.

After you have entered a name (and perhaps a description), click the Details tab. The fields on this tab are

Let the SAS System assign the library	specifies who is responsible for assigning this library—you or SAS/Warehouse Administrator.
Libref	specifies a libref for the library being defined.
Path	specifies a physical path for the library being defined. For details about specifying pathnames, see "Specifying Physical Pathnames in SAS/Warehouse Administrator" on page 66.



[✓ Let the SAS System assign the library] The user will pre-assign the library

Libref: remdata

Engine:

Options:

Path: ./toy_unx_srcdata

The **Let the SAS System assign the library** option is selected in the Details tab above.

The **Libref** field contains a libref appropriate for a remote data source library: **remdata**.

Note: Do not specify any values in the **Engine** field or the **Options** field on this tab. All remote host information is provided in a separate host definition. △

The **Path** field above contains a relative pathname that is appropriate for a SAS library on a UNIX machine: **./toy_unx_srcdata**.

Note: In order for the remote library definition to work properly, the pathname entered in the **Path** field must resolve after connection to the remote computer is established. △

In the metadata for data stores and other objects, you will specify both the library definition and the host definition for the computer where the library resides. A host definition includes a reference to a SAS/CONNECT script. This script starts a session in a particular directory on the remote computer. The library pathname entered in the **Path** field must resolve from the start directory on the remote computer, as specified by the host definition you plan to use with this remote library definition.

For example, suppose that you plan to use a host definition called **unix_1** with the **Remote Src Data** library definition in the current example. Suppose also that the **unix_1** definition included a reference to a SAS/CONNECT script, which started a SAS session on a UNIX computer in the /user/admin1 directory. In that case, the pathname entered in the **Path** field above must resolve from the /user/admin1 directory on the target UNIX machine.

After specifying the values above, click $\boxed{\text{OK}}$. The new library definition is added to the list of libraries in the Define Items Used Globally window.

Example: Creating a SAS/ACCESS LIBNAME Definition

This example summarizes how to create a metadata record for a SAS/ACCESS LIBNAME definition, which is a special SAS library that can be used to extract source data in DBMS format or to create warehouse data stores in a DBMS.

SAS/Warehouse Administrator uses a SAS/ACCESS 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 SAS/ACCESS 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 SAS/ACCESS LIBNAME definition also specifies a DBMS connection profile, which includes the DBMS user ID, password, server name, and other connection information used to access the DBMS. These options are passed to DBMS client software, which actually makes the connection to the DBMS.

SAS/ACCESS LIBNAME definitions can be used 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 the *SAS Language Reference: Dictionary*.

Preparing to Create a SAS/ACCESS LIBNAME Definition

Before you create a SAS/ACCESS LIBNAME definition, make the following preparations:

- □ Verify that you have the software that you need, as described in “Access to Data In DBMS Format” on page 28.
- □ Identify the target DBMS—the database to be accessed with the SAS/ACCESS LIBNAME definition. Identify the DBMS username and password you will need to make the connection.
- □ Install and configure the DBMS client software on the computer where the SAS/ACCESS LIBNAME statement will be executed.
- □ In the DBMS client, define a connection for the target DBMS.
- □ In SAS/Warehouse Administrator, create a DBMS connection profile for the target DBMS. For details, see “Example: Creating a Connection Profile for a SAS/ ACCESS LIBNAME Definition” on page 95.

Test SAS/ACCESS Engine Support

SAS/Warehouse Administrator uses a SAS/ACCESS LIBNAME definition to generate a SAS/ACCESS LIBNAME statement. In order for such a statement to work, you must have SAS/ACCESS software, a DBMS client for the target DBMS, and a DBMS connection profile for the target DBMS. To verify that these elements are working, you might want to manually submit a LIBNAME statement similar to the one that SAS/ Warehouse Administrator will generate for the SAS/ACCESS LIBNAME definition that you are creating. If your manually submitted statement is successful in connecting to the target DBMS, the statement generated by SAS/Warehouse Administrator will work also.

For example, suppose that you have installed the relevant software and have gathered the following information:

Target DBMS	**`Oracle`**
DBMS user name	**`admin1`**
DBMS password	**`ad1min`**
DBMS connection name	**`V2o7223.world`**

On the computer where the SAS/ACCESS LIBNAME statement will be executed, you could then submit a LIBNAME statement such as the following in the SAS Program Editor:

```
libname mydb oracle user=admin1 pass=ad1min path='V2o7223.world';
```

The options are specific to the DBMS to which you are connecting. For details about options, see the appropriate chapter in *SAS/ACCESS for Relational Databases: Reference*. Some of the options in the LIBNAME statement above are specified in the

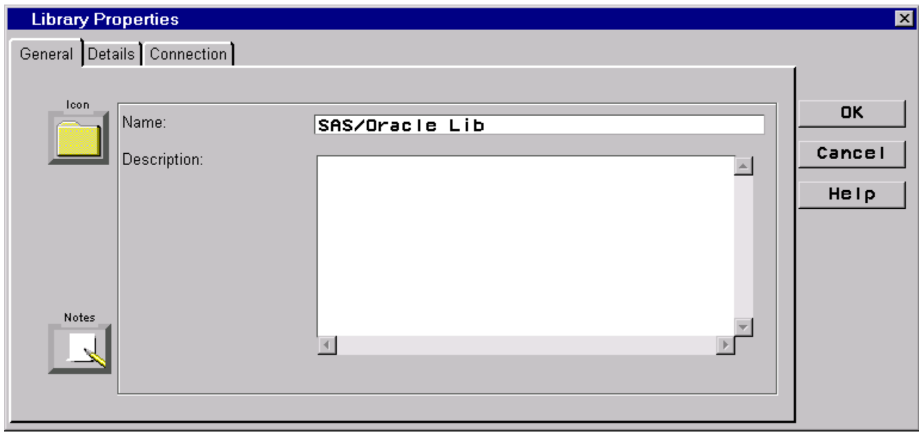
SAS/ACCESS LIBNAME definition. Others are specified in the DBMS connection profile for the target DBMS.

Define SAS/ACCESS LIBNAME Properties

Display the Define Items Used Globally Window, as described in "Using the Define Items Used Globally Window" on page 76.

In the Type panel of the Define Items Used Globally window, click the button beside **SAS Libraries**. Click Add at the bottom left of the window. A Library Properties window for the library displays for you to enter the appropriate information as follows:

Name	enter a display name for the SAS library you are creating.

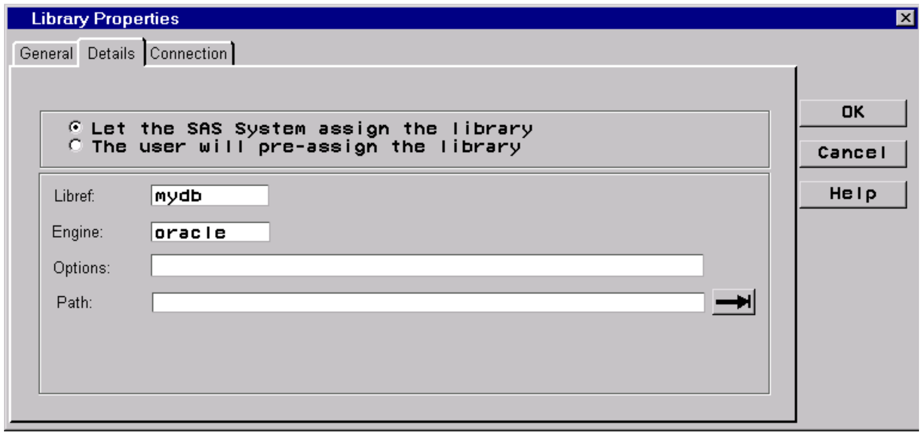


Name: SAS/Oracle Lib

The **Name** field contains a display name that is appropriate for a SAS/ACCESS LIBNAME definition for an Oracle database: **SAS/Oracle Lib**.

After you have entered a name (and perhaps a description), click the Details tab. The fields on this tab are

Let the SAS System assign the library	specifies who is responsible for assigning this library—you or SAS/Warehouse Administrator.
Libref	specifies a libref for the library being defined.
Engine	specifies a SAS/ACCESS engine for the library being defined.



[✓ Let the SAS System assign the library]
The user will pre-assign the library

Libref: mydb

Engine: oracle

The **`Let the SAS System assign the library`** option is selected in the Details tab above.

Selecting **`Let the SAS System assign the library`** lets SAS/Warehouse Administrator assign this library when you execute a Job that includes a reference to this library definition. Do not select this option if you want to control when this library is assigned.

The **`Libref`** field above contains the libref **`mydb`**.

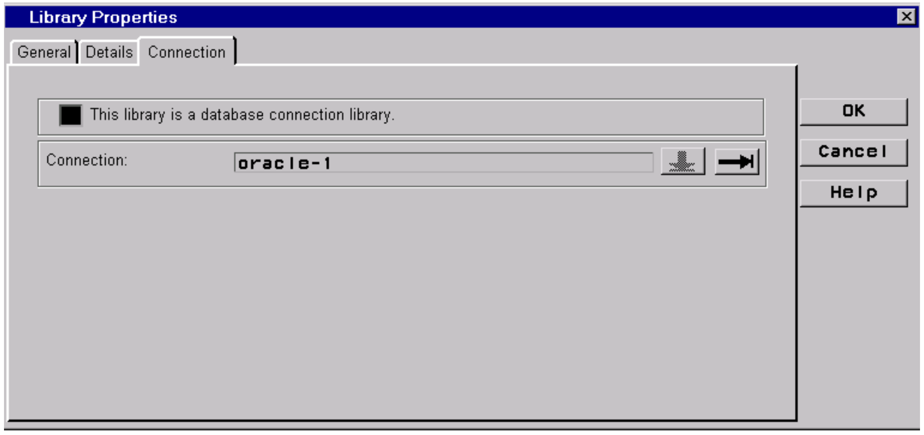
The **`Engine`** field above specifies that the **`oracle`** SAS/ACCESS engine will be used to access this library.

The **`Options`** and **`Path`** fields are left blank.

Note: If you are creating a SAS/ACCESS LIBNAME definition, the DBMS user ID, password, and server name should not be entered here in the **`Options`** field. That information should be specified as part of the DBMS connection profile that you will specify on the Connection tab. △

After specifying the previous values, click the Connection tab. The fields on this tab are

`This library is a database connection library`	specifies that the library you are defining requires a DBMS connection profile.
`Connection`	specifies a DBMS connection profile.



This library is a database connection library.

Connection: oracle-1

The **`This library is a database connection library`** option is selected in the Connection tab above.

In the **`Connection`** field, an appropriate DBMS connection profile was selected using the down arrow: **`oracle-1`**. For details about connection profiles, see "DBMS Connection Profiles" on page 95.

After specifying the previous values, click OK. The new library definition is added to the list of libraries in the Define Items Used Globally window.

Example: Creating a Jobs Information Library

If you want SAS/Warehouse Administrator to generate code that will execute a Job at a future date and time, you must define one or more Jobs Information libraries. This example describes how to create a remote Jobs Information library under the control of a SAS/SHARE server, which is the recommended configuration for a production data warehouse in which Jobs are scheduled and tracked through SAS/Warehouse Administrator.

A Jobs Information library is a SAS library that contains status information for Jobs that have been scheduled through the Job Properties window in SAS/Warehouse Administrator. If job tracking is enabled for a given Job, when the Job executes, it will update its status in the appropriate Jobs Information library. The Job Viewer window reads the Jobs Information library to display information about Jobs that have been submitted.

Preparing to Create a Jobs Information Library

The basic steps for creating a Jobs Information library definition are the same as for any SAS library definition: open the appropriate Warehouse Environment and add a library definition from the Define Items Used Globally window. However, there are some special considerations for Jobs Information libraries:

- □ Jobs Information libraries are for internal use by SAS/Warehouse Administrator. Accordingly, they are often stored on the same host where the metadata repositories are stored. In "Metadata Host Configuration" on page 23, see the configurations that support Job scheduling and tracking.
- □ The physical path for each Jobs Information library must exist.
- □ For a production data warehouse in which Jobs are scheduled and tracked through SAS/Warehouse Administrator, it is strongly recommended that the Jobs Information libraries be placed under the control of a SAS/SHARE server. For details, see "Jobs Information Libraries and SAS/SHARE Software" on page 24.

Note: For a non-production data warehouse, or for a data warehouse where Jobs are not tracked through SAS/Warehouse Administrator, it is not necessary to put the relevant Jobs Information libraries under the control of a SAS/SHARE server. In that case, you can simply add a local or remote library definition, then make that library the Jobs Information library for a given Environment or Data Warehouse, as described in "Registering Jobs Information Libraries" on page 306. △

SAS/SHARE Server Preparation

On the SAS/SHARE server host,

- □ create a directory for the Jobs information library
- □ assign a SAS libref to this directory in such a way that the library is under the control of the SAS/SHARE server.

For example, you might create the .\Project-2_jobinfo_1 directory on the SAS/SHARE server host, then assign the libref joblib_1 to that directory so that it is under the control of the SAS/SHARE server.

Note: If you use a relative pathname such as .\Project-2_jobinfo_1 for the Jobs Information library, be sure that the SAS/SHARE server can resolve the pathname. △

SAS/Warehouse Administrator Client Preparation

Verify that your local SAS session can access the remote library that will ultimately become the Jobs Information library that you are creating. One way to do that is to submit a LIBNAME statement for that library. Here is an example of such a statement:

```
libname joblib_1 server=host2.dwshare;
```

where **`joblib_1`** is the libref for the Jobs Information library on both the SAS/Warehouse Administrator host and the remote SAS/SHARE server host, **`host2`** is the name of the remote host, and **`dwshare`** is the name of the SAS/SHARE server.

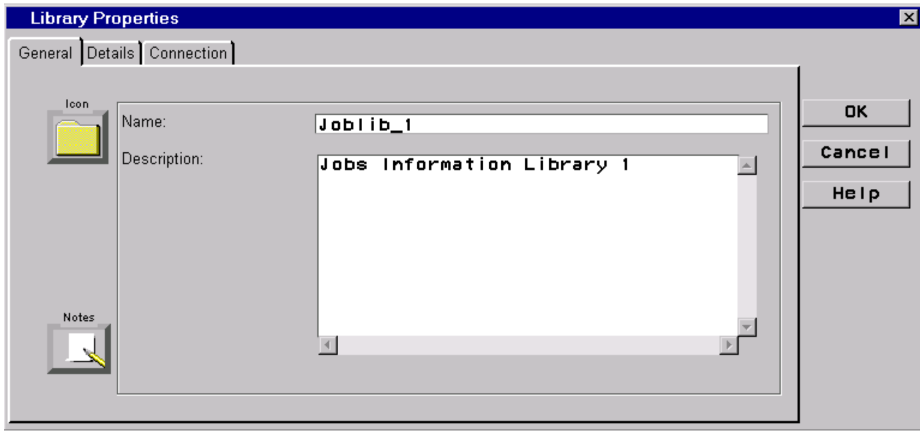
If you can successfully execute such a statement, then SAS/Warehouse Administrator can successfully generate code for the remote Jobs Information library.

Define Library Properties

Display the Define Items Used Globally Window, as described in “Using the Define Items Used Globally Window” on page 76.

In the Type panel of the Define Items Used Globally window, click the button beside **`SAS Libraries`**. Click Add at the bottom left of the window. A Library Properties window for the library displays for you to enter the appropriate information as follows:

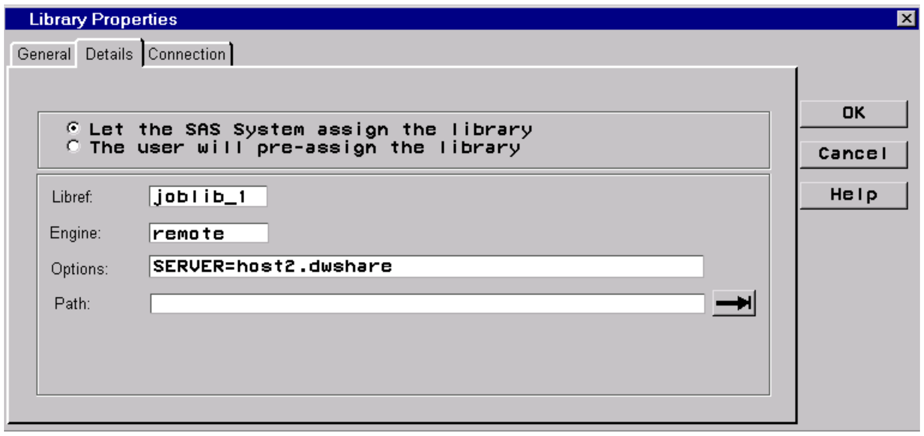
`Name`	specifies a display name for the SAS library you are creating, such as **`Joblib_1`**.



Name: Joblib_1
Description: Jobs Information Library 1

After you have entered a name (and perhaps a description), click the Details tab. The fields on this tab are

`Let the SAS System assign the library`	specifies who is responsible for assigning this library—you or SAS/Warehouse Administrator.
`Libref`	specifies a libref for the library being defined.
`Engine`	specifies a SAS engine appropriate for the library being defined.
`Options`	specifies any LIBNAME statement options appropriate for the library being defined.



[✓ Let the SAS System assign the library] The user will pre-assign the library

Libref: joblib_1
Engine: remote
Options: SERVER=host2.dwshare
Path:

On the Details tab, the **Let the SAS System assign the library** option is selected.

The **Libref** field contains a local libref appropriate for a Jobs Information library: **joblib_1**. The current example assumes that the libref for the Jobs Information library is **joblib_1** on both the SAS/Warehouse Administrator host and the remote SAS/SHARE server host.

The **Engine** field specifies the **remote** engine, because the library in our example is on a host that is remote to SAS/Warehouse Administrator.

The **Options** field specifies the SERVER= option required to access the Jobs Information library, where **host2** is the name of the remote host, and **dwshare** is the name of the SAS/SHARE server.

After specifying values in these fields, click OK. The new library definition is added to the list of libraries in the Define Items Used Globally window.

After you have defined the library, you can make that library the Jobs Information library for a given Environment or Data Warehouse, as described in "Registering Jobs Information Libraries" on page 306.

Host Definitions

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 the current Environment. They are required in order to access source data and to load warehouse data stores.

Preparing to Create Host Definitions

Here is one way to identify the host definitions you need for a given project:

- Review the host configuration in your project plan.
- Create a *local* host definition for each computer where SAS/Warehouse Administrator is installed.
- Create a *remote* host definition for each computer that is remote to SAS/Warehouse Administrator where a data store resides, where a process or Job will execute, or where a process will send its output—if SAS/Warehouse Administrator will generate code for that object.

Local and Remote Host Definitions

Host definitions include a **Warehouse Jobs Are** field. This field determines whether SAS/Warehouse Administrator will generate SAS/CONNECT statements for the object whose metadata includes the host definition. The possible values for this field are

Local to this Host	SAS/Warehouse Administrator generates code for the object as if it were local to SAS/Warehouse Administrator. (The code does not include SAS/CONNECT statements.)
Remote to this Host	SAS/Warehouse Administrator generates code for the object as if it were remote to SAS/Warehouse Administrator. (The code includes SAS/CONNECT statements.)

For example, suppose that you want SAS/Warehouse Administrator to generate code for a data store that resides on the SAS/Warehouse Administrator host. The metadata for this library would include a local host definition because SAS/CONNECT is not required to execute the code for this object.

To take another example, suppose that you want SAS/Warehouse Administrator to generate code to access a remote DBMS through a local SAS/ACCESS LIBNAME statement. The metadata for the SAS/ACCESS LIBNAME statement would include a local host definition because SAS/CONNECT is not required to execute a local SAS/ACCESS LIBNAME statement. (In this case, the DBMS is accessed as if it were a local SAS library, for the most part. The remote connection is handled transparently by the DBMS client software.)

On the other hand, suppose that you want SAS/Warehouse Administrator to generate code for a data store that resides on a host remote to SAS/Warehouse Administrator. The metadata for this object would include a remote host definition because SAS/CONNECT is required to access the remote host where the data store resides.

Example: Adding a Local Host Definition

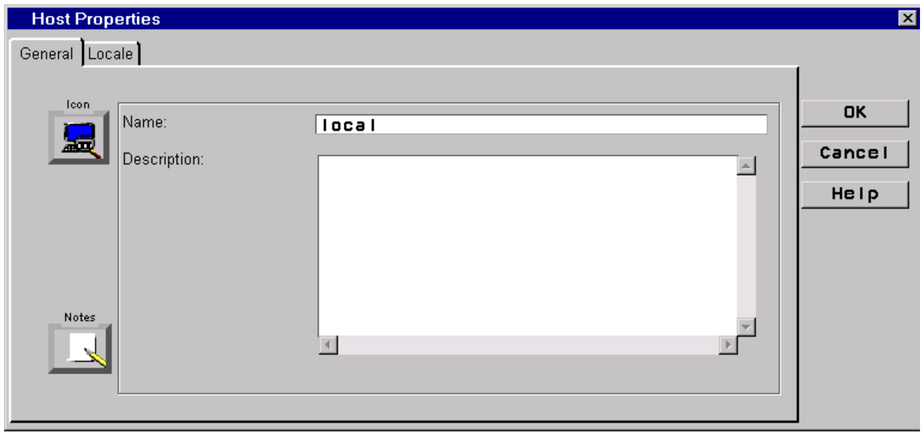
This example summarizes how to add a local host definition—a host definition with a **`Local to this Host`** value in its **`Warehouse Jobs Are`** field. Typically, you will create a local host definition for each computer where SAS/Warehouse Administrator is installed. The appropriate Warehouse Environment is assumed to exist.

Define Host Properties

Display the Define Items Used Globally window, as described in "Using the Define Items Used Globally Window" on page 76.

In the Type panel of the Define Items Used Globally window, click the button beside **`Hosts`**. Click Add at the bottom left of the window. A Host Properties window for the host displays for you to enter the appropriate information as follows:

`Name`	specifies a display name for the host definition that you are creating.



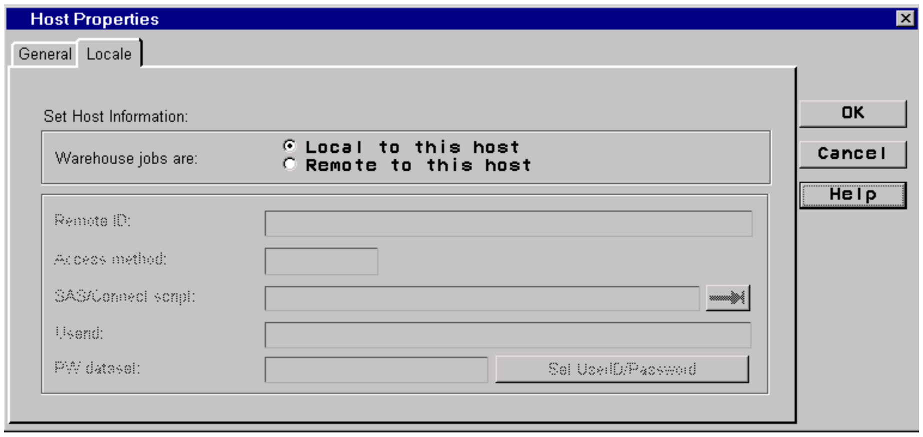
Name: local
Description:

The **`Name`** field contains a display name that is appropriate for a local host: **`local`**. If your site has multiple SAS/Warehouse Administrator hosts, you might want to give each local host a different display name.

After you have entered a name (and perhaps a description), click the Locale tab. The fields on this tab are

`Warehouse jobs are`	specifies whether the host being defined is local or remote to SAS/ Warehouse Administrator—for code generation purposes.

If you select **Local to this host**, when you incorporate this host definition in the metadata for a data store, process, or Job, SAS/Warehouse Administrator will generate code for that object as if it is local to SAS/Warehouse Administrator (the code will not include SAS/CONNECT statements).



Set Host Information:

Warehouse jobs are: [✓ Local to this host] Remote to this host

Remote ID:

Access method:

SAS/Connect script:

Userid:

PW dataset:

Set UserID/Password

In the **Warehouse jobs are** field, the **Local to this host** option is selected.

After specifying values in this field, click OK. The new host definition is added to the list of hosts in the Define Items Used Globally window.

Example: Adding a Remote Host Definition

This example summarizes how to add a remote host definition—a host definition with a **Remote to this Host** value in its **Warehouse Jobs Are** field. Typically, you will create a remote host definition for each computer that is remote to SAS/Warehouse Administrator where a data store resides, where a process or Job will execute, or where a process will send its output—if SAS/Warehouse Administrator will generate code for that object.

The appropriate Warehouse Environment is assumed to exist.

Preparing to Create a Remote Host Definition

Before you create a remote host definition, gather the following information about the host:

Access method	Example: **TCP/IP**
Remote host ID	Example: **unixone.mycompany.com**
Remote User ID	Example: **admindw**
Remote Password	Example: **addwmin**
Remote SAS Version *	Example: **SAS Version 6.12**

* Keep in mind that a remote host definition is used to generate the code for a SAS/CONNECT session between two computers, one of which is the SAS/Warehouse Administrator host. Base SAS must be installed on both machines.

In addition to gathering the information above, you should also do the following tasks:

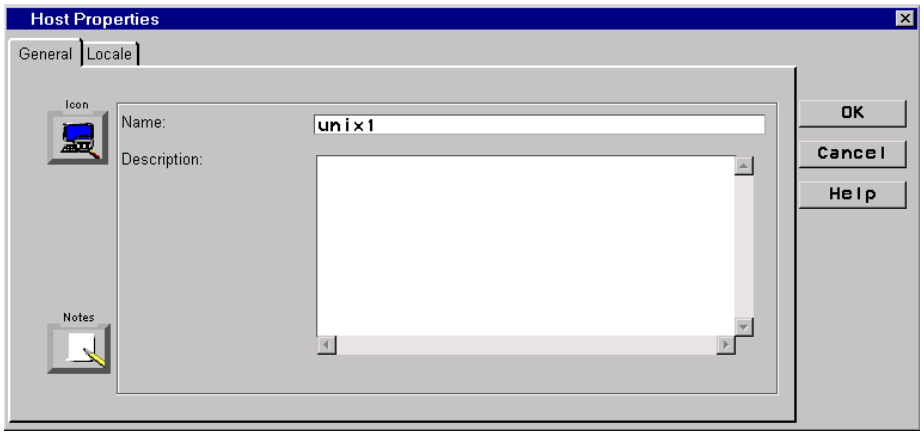
- □ Complete the relevant setup tasks for remote hosts, as described in "Additional Setup for Remote Hosts" on page 92.
- □ Test the SAS/CONNECT script that you will specify in the host definition that you are creating.

Define Host Properties

Display the Define Items Used Globally window, as described in "Using the Define Items Used Globally Window" on page 76.

In the Type panel of the Define Items Used Globally window, click the button beside **`Hosts`**. Click Add at the bottom left of the window. A Host Properties window for the host displays for you to enter the appropriate information as follows:

`Name`	specifies a display name for the host definition that you are creating.



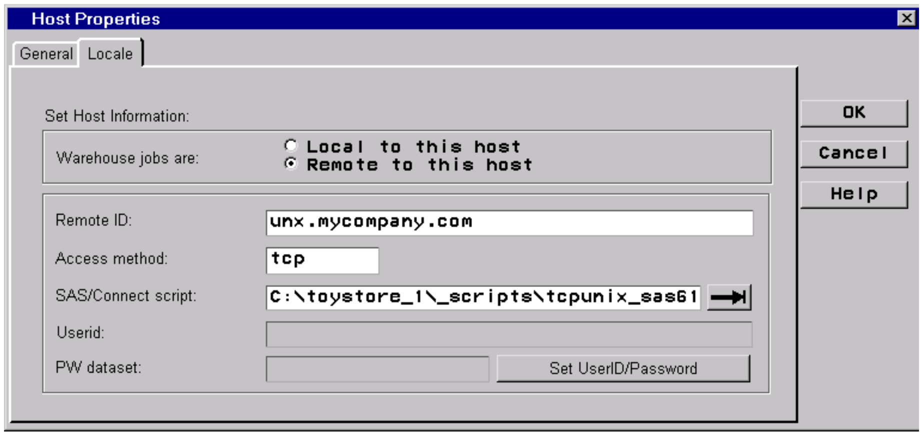
Name: unix1
Description:

The **`Name`** field contains a display name that is appropriate for a remote UNIX host: **`unix1`**.

After you have entered a name (and perhaps a description), click the Locale tab. The fields on this tab are

`Warehouse jobs are`	specifies whether the host being defined is local or remote to SAS/Warehouse Administrator—for code generation purposes. If you select **`Remote to this host`** when you incorporate this host definition in the metadata for a data store, process, or Job, SAS/Warehouse Administrator will generate code for that object as though it were remote to SAS/Warehouse Administrator (the code will include SAS/CONNECT statements).
`Remote ID`	specifies the address needed to access the remote host—as appropriate for the access method you specify in the **`Access method`** field.
`Access method`	specifies the communications access method that you want to use to access this host.
`SAS/CONNECT script`	specifies the full pathname of the logon script for this host—if a script is required for your access method. For details about creating the SAS/CONNECT script that is referenced in this field, see "Creating Scripts for Accessing Remote Hosts" on page 92.

Userid	displays the user ID to be supplied to the script specified in the **SAS/ CONNECT script** field. The user ID is entered from the window that is displayed when you click Set UserID/Password.
PW dataset	displays the encrypted password data set used to store the user ID and password to be supplied to the script specified in the **SAS/ CONNECT script** field. The data set name (such as SASUSER.ENCRYPT) is entered from the window that is displayed when you click Set UserID /Password.



Set Host Information:

Warehouse jobs are: ( ) Local to this host [✓ Remote to this host]

Remote ID: unx.mycompany.com

Access method: tcp

SAS/Connect script: C:\toystore_1_scripts\tcpunix_sas61

Userid:

PW dataset:

In the **Warehouse jobs are** field, the **Remote to this host** option is selected.

In the **Remote ID** field, a domain name for the remote host is specified: **unixone.mycompany.com**. This is a name appropriate for the **tcp** (TCP/IP) protocol specified in the **Access method** field.

In the current example, the remote host has Release 6.12 of SAS installed. Accordingly, in the **SAS/CONNECT script** field, the full pathname for a script appropriate for Release 6.12 is specified: **C:\toystore_1_scripts\tcpunx_sas612.scr**.

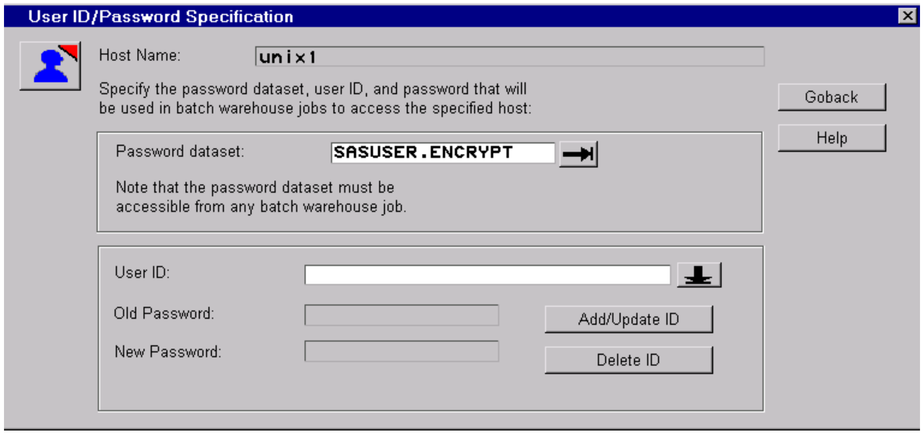
If the script specified in the **SAS/CONNECT script** field will not prompt for a user ID and password, or if you write your own code to provide the user ID and password to the script when it runs in batch mode, then you are finished with this host definition. Click OK. The new host definition is added to the list of hosts in the Define Items Used Globally window.

On the other hand, if the script specified in the **SAS/CONNECT script** field will prompt for a user ID and password, and you want SAS/Warehouse Administrator to supply them, then click Set UserID /Password to display the User ID/Password Specification window. The fields in this window are

Password dataset	specifies the encrypted data set where the user ID and password for the remote host are stored. The default is SASUSER.ENCRYPT. This data set must be accessible during batch jobs and interactive SAS/Warehouse Administrator sessions.
User ID	specifies the user ID needed to access the remote host. SAS/Warehouse Administrator stores the user ID and password in the encrypted data set that is specified in the **Password dataset** field.
Old Password	specifies a password that you want to use to access the remote host.
New Password	specifies the new password that you want to use to access the remote host.

Note: The password that is specified in the User ID/Password Specification window cannot contain delimiters such as: . **< (+ & ! $ *) ; - / , % |**. △

The password specified in this window will be read by a SAS macro function such as **%qscan**. These macro functions cannot read passwords with the delimiters listed above. If the password cannot be read, the remote host login script that requires this password might fail.


User ID/Password Specification
Host Name:
unix1
Specify the password dataset, user ID, and password that will be used in batch warehouse jobs to access the specified host:
Password dataset:
SASUSER.ENCRYPT
Note that the password dataset must be accessible from any batch warehouse job.
User ID:
Old Password:
New Password:
Add/Update ID
Delete ID
Goback
Help

In the User ID/Password Specification window, you specify the user ID and password that SAS/Warehouse Administrator will provide to the SAS/CONNECT script that is specified in the **SAS/CONNECT script** field on the Locale tab. You will also specify the data set where this information is stored.

In the **Password dataset** field, the default data set is selected: **SASUSER.ENCRYPT**.

To enter a new user ID and password for this host definition:

1 Enter the user ID in the **User ID** field and press RETURN.

2 Enter the password and press RETURN.

3 Enter the password a second time to verify it and press RETURN.

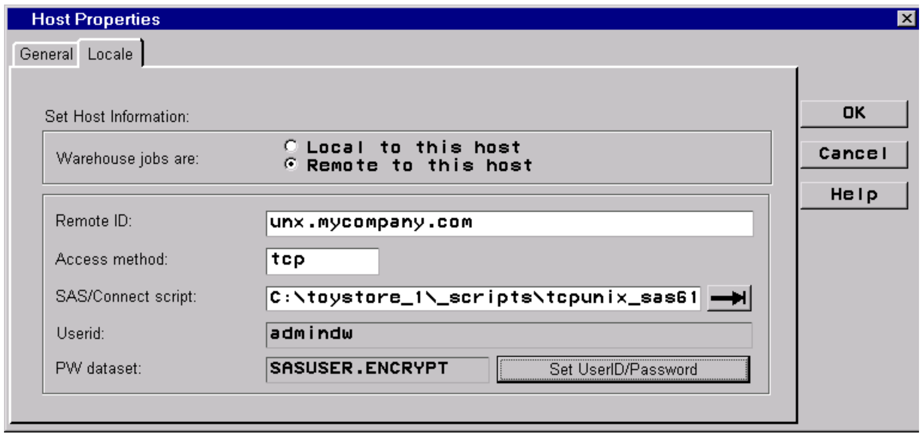
4 Click Add/Update to save the specified user ID and password in the password data set.

   Note: You must click Add/Update ID to save the specified user ID and password in the password data set. Otherwise your changes will be lost, and the remote host login script that requires this user ID and password might fail. △

5 Click Go Back.

   You will be returned to the Locale tab.

After adding security information through the User ID/Password Specification window, the **Userid** and **PW dataset** fields on the Locale tab are updated, as shown in the next display:


Host Properties

General | Locale

Set Host Information:

Warehouse jobs are: Local to this host [✓ Remote to this host]

Remote ID: unx.mycompany.com

Access method: tcp

SAS/Connect script: C:\toystore_1_scripts\tcpunix_sas61

Userid: admindw

PW dataset: SASUSER.ENCRYPT

Verify that the **Userid** and **PW dataset** fields reflect the entries made in the User ID/Password Specification window. If so, you are finished with this host definition.

Click OK. The new host definition is added to the list of hosts in the Define Items Used Globally window.

Additional Setup for Remote Hosts

If you have created any remote host definitions in the Host Properties window, you must do some additional setup tasks, or the connections specified in the remote host definitions might fail.

Creating Scripts for Accessing Remote Hosts

The metadata for a remote host definition includes a reference to a SAS/CONNECT script if a script is required for your access method. You can write your own scripts, or you can copy and modify the example scripts that are shipped with SAS/CONNECT software. The example scripts have an ***.scr** extension and are typically found in a subdirectory under the SAS/CONNECT directory.

Verify that the SAS/CONNECT script that is referenced by a host definition

- starts the correct version of SAS, with the appropriate options
- does not prompt you for a user ID and password.

Scripts for accessing remote hosts should be accessible from batch sessions as well as interactive sessions. Accordingly, they cannot prompt you for user ID and password information.

Usually, your script will prompt you for your user ID and password, using code such as the following:

```
input 'Userid?';
waitfor 'ENTER PASSWORD', 60 seconds : nolog;

input nodisplay 'Password?';
```

To allow the script to run in batch, you should replace this code with the following:

```
type "%qscan(%superq(xxxxsec),1,.)";
type LF;
waitfor 'ENTER PASSWORD', 60 seconds : nolog;

type "%qscan(%superq(xxxxsec),2,.)";
type LF;
```

In the previous code, *xxxx* is an abbreviation for the access method. For example, if you are using APPC, you would specify the string **`appcsec`** as the macro variable. If you are using TCP, you would specify the string **`tcpsec`**.

Note: If you have created a remote host definition that includes user ID and password information stored in an encrypted data set, you must enable GETUSRPW macro support on the host where the login script runs, as described below. △

Setup Required for GETUSRPW Macro

SAS/Warehouse Administrator provides the macro GETUSRPW to retrieve the user ID and password from the encrypted data set specified on the User ID/Password Specification window. The main purpose of the macro GETUSRPW is to appropriately set the *xxxx*SEC macro variable (where *xxxx* is the access method) to USERID.PASSWORD.

If you specified a script in the **`SAS/Connect script`** field of the Host Properties window, and you had SAS/Warehouse Administrator generate the code to provide a user ID and password to the script, you must enable the GETUSRPW macro support on the platform where the script will run. For example, if you specified a script that will run on the same Windows NT machine where SAS/Warehouse Administrator is installed, you must enable the GETUSRPW macro support on that NT machine. Otherwise, if you specified a script that will run on a mainframe, you must enable the GETUSRPW macro support on that mainframe.

The following sections explain how to get the GETUSRPW macro to work on the appropriate platform.

GETUSRPW Support Local to SAS/Warehouse Administrator

For PC platforms, place the following code in your SAS autoexec file:

```
libname saswa '!sasroot\whouse\sashelp';
filename getusrpw catalog 'saswa.dwport';
options sasautos=(getusrpw sasautos);
```

For UNIX platforms, place the following code in your SAS autoexec file:

```
libname saswa '!sasroot/sashelp';
filename getusrpw catalog 'saswa.dwport';
options sasautos=(getusrpw sasautos);
```

Note: The SASWA libref points to the location of SAS/Warehouse Administrator's DWPORT catalog. The example above shows the default location. If you install SASWA somewhere other than the default location, you should make the changes appropriately. △

Restart SAS with the new SAS autoexec file.

GETUSRPW Support Remote to SAS/Warehouse Administrator

Create a SAS Data library on the target platform where you need GETUSRPW support. The location of the library can be whatever you choose. This will be the location of the GETUSRPW components.

Allocate the SAS Data library with a library reference (libref) of SASWA.

Upload the following catalog entries from the platform where SAS/Warehouse Administrator is installed to the target platform where GETUSRPW support is needed:

ENCDEC.CLASS, ENCPWUSR.CLASS, ENCDEC.SCL, ENCPWUSR.SCL, GETUSRPW.SCL, GETUSRPW.SOURCE.

Also, you will need to move the ENCRYPT data set to the target platform. The GETUSRPW.SOURCE entry assumes that this data set is named SASUSER.ENCRYPT. If you give it a different name, you will need to update the GETUSRPW.SOURCE entry.

Here is sample code for uploading to OS/390 (MVS) (with a new allocation of the SASWA library):

```
options remote=sdcmvs;
filename rlink '!SASROOT\connect\saslink\tcptso.scr';
signon;

rsubmit;
libname saswa 'library name' disp=(new,catlg) space=(cyl,(3,1));
proc upload incat=sashelp.dwport outcat=saswa.dwport status=no;
  select
   ENCDEC.CLASS
   ENCPWUSR.CLASS
   ENCDEC.SCL
   ENCPWUSR.SCL
   GETUSRPW.SCL
   GETUSRPW.SOURCE;

  proc upload in=sasuser out=sasuser status=no;
    select encrypt;
  run;
endrsubmit;
signoff;
```

Make the GETUSRPW macro available to remote Jobs. The best way to do this is by making GETUSRPW.SOURCE a part of the Autocall Macro Facility search path. The search path is defined by the SAS option SASAUTOS. Here are two examples: one for Release 6.11 of SAS and later, the other for Release 6.09 of SAS (MVS, CMS).

For Release 6.11 of SAS and later, add the following to your SAS autoexec file:

```
libname saswa 'user defined location';
filename getusrpw catalog 'saswa.dwport';
options sasautos=(getusrpw sasautos);
```

For Release 6.09 of SAS (MVS, CMS), start Interactive SAS, use the LIBNAME statement to allocate the libref SASWA to point to the library that you created. On the SAS command line, type **`copy saswa.dwport.getusrpw.source`**.

Save the contents of the program editor to a PDS member on OS/390 (MVS) or a maclib on CMS with the name of **`getusrpw`**. Here is an example of the FILE command on OS/390 (MVS):

```
file 'userid.my.pds(getusrpw)'
```

Add the following to your SAS autoexec file:

```
libname saswa 'user defined location';
filename getusrpw 'PDS or Maclib name';
options sasautos=(getusrpw sasautos);
```

Note: The libref SASWA should point to the library created in the first step of the process that contains the macro GETUSRPW components. △

Make sure that each invocation of SAS points to this new SAS autoexec file.

DBMS Connection Profiles

A DBMS connection profile is 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 SAS/ACCESS LIBNAME definitions in the current 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.

Preparing to Create DBMS Connection Profiles

In SAS/Warehouse Administrator, you can create two kinds of connection profiles:

- □ connection profiles that specify SAS/ACCESS LIBNAME statement options and are included in the metadata for SAS/ACCESS LIBNAME definitions
- □ connection profiles that specify SQL Pass-Through statement options and DBLOAD options that are included directly in the metadata for DBMS data stores—data stores with Load processes that are similar to those provided by SAS/Warehouse Administrator Release 1.x.

Before you create a connection profile, determine how it will be used. This will help you determine what kinds of options you will specify in the profile.

Example: Creating a Connection Profile for a SAS/ACCESS LIBNAME Definition

This example summarizes how to create a connection profile that will be included in the metadata for a SAS/ACCESS LIBNAME definition.

Identify DBMS Login Information

Before you create a connection profile that will be used in a SAS/ACCESS LIBNAME definition, gather the following information:

Target DBMS	Example: **`Oracle`**
DBMS user name	Example: **`admin1`**
DBMS password	Example: **`ad1min`**
DBMS connection name	Example: **`V2o7223.world`**

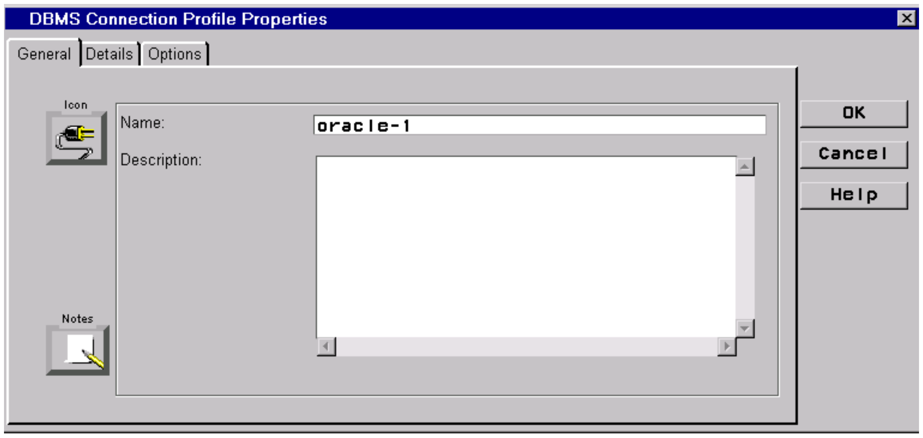
Define Connection Properties

Display the Define Items Used Globally window, as described in “Using the Define Items Used Globally Window” on page 76.

In the Type panel of the Define Items Used Globally window, click the button beside **`DBMS Connections`**. Click $\boxed{\text{Add}}$ at the bottom left of the window. A DBMS Connection

Profile Properties window for the connection displays for you to enter the appropriate information as follows:

`Name`	enter a name for this connection profile.

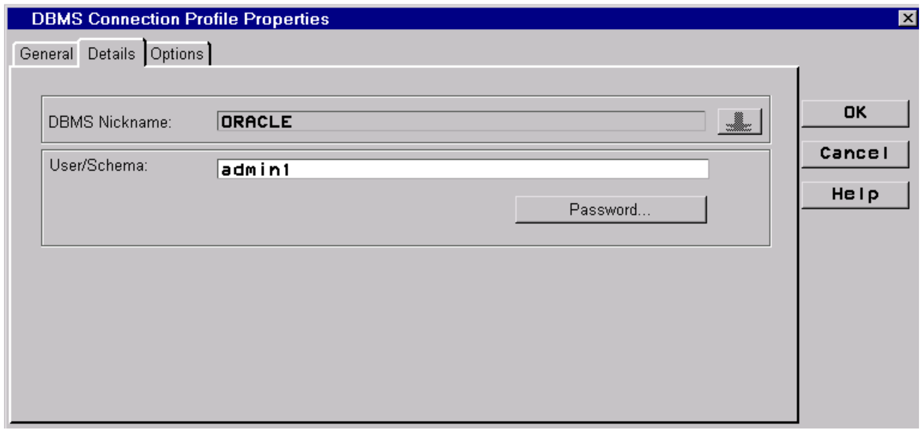


Name: oracle-1
Description:

In the **`Name`** field, **`oracle-1`** is specified.

After you have specified a name (and possibly, a description), click the Details tab. The fields on this tab are

`DBMS Nickname`	select the target DBMS from the drop-down list.
`User/Schema`	specify the name that is used to log in to the target DBMS.



DBMS Nickname: ORACLE
User/Schema: admin1

In the **`DBMS Nickname`** field, **`Oracle`** was selected from a list of DBMS nicknames known to SAS.

In the **`User/Schema`** field, **`admin1`** is the name that is used to log in to the DBMS in our example.

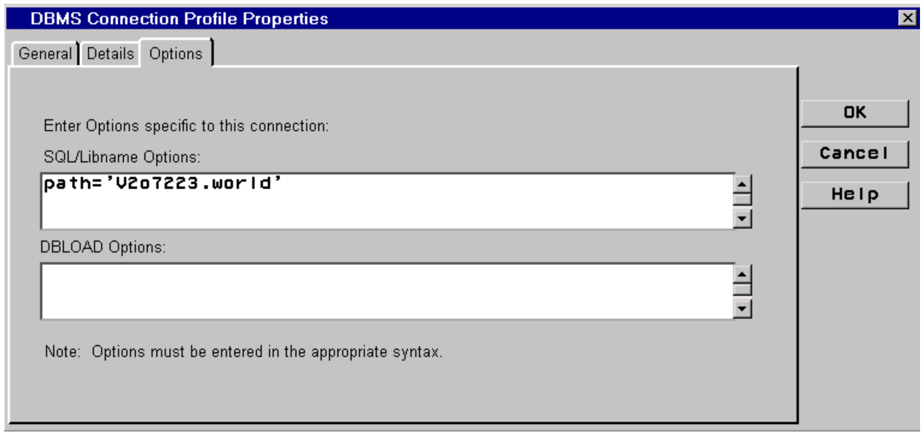
After you have entered the **`User/Schema`**, click Password to define the password used to log in to the target DBMS. In the Password window, enter the password, press RETURN, then enter the password again to verify it. You will be returned to the Details tab.

Click the Options tab. The fields on this tab are

`SQL/LIBNAME options`	specify SAS/ACCESS LIBNAME statement options or SQL Pass-Through statement options for the current connection profile.

If you are defining a DBMS connection for a SAS/ACCESS LIBNAME definition, specify SAS/ACCESS LIBNAME statement options in this field. At a minimum, specify the DBMS server name required to connect to the target DBMS.

The options are specific to the DBMS to which you are connecting. For details about options, see the appropriate chapter in *SAS/ ACCESS for Relational Databases: Reference*.



Enter Options specific to this connection:

SQL/Libname Options:

```
path='V2o7223.world'
```

DBLOAD Options:

Note: Options must be entered in the appropriate syntax.

In the current example, the appropriate value for the **SQL/LIBNAME Options** field is a PATH= option that specifies the DBMS server name—with the server name enclosed in single quotation marks: **path='V2o7223.world'**. Use the same syntax that you would use in a SAS/ACCESS LIBNAME statement, such as the one in "Test SAS/ ACCESS Engine Support" on page 81.

Leave the **DBLOAD Options** field blank if you are defining a DBMS connection for a SAS/ACCESS LIBNAME definition.

You have now entered all of the metadata required for this profile. Click OK to save the profile. The new profile is added to the list of connections in the Define Items Used Globally window.

Contact Records

A contact record is 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 formulated by the owner in accordance with established policy.

Contact records can be included in the metadata for groups, data stores, processes, Jobs, and other objects in the current 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 responsible for a given warehouse element.

Preparing to Create Contact Records

- ☐ Review the SAS/Warehouse Administrator groups and data stores in your project plan.

- □ Identify the owners and administrators of these groups and data stores.
- □ Record the appropriate contact information for each person.

Example: Creating a Contact Record

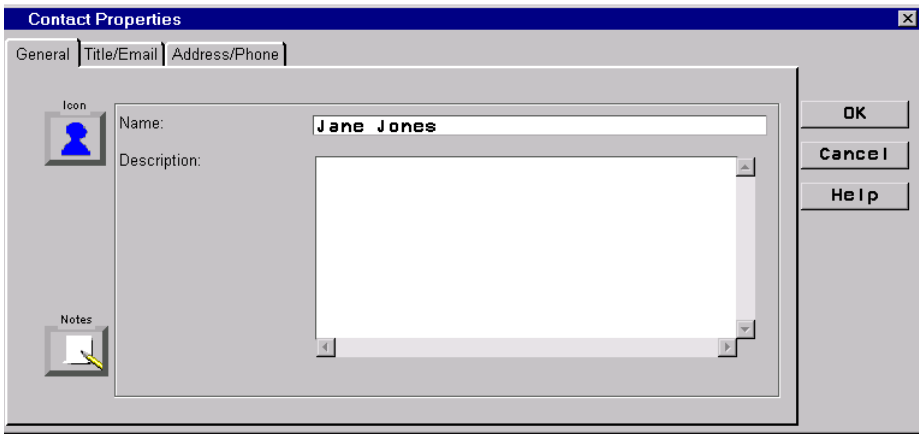
This topic summarizes how to create a contact record. The appropriate Warehouse Environment is assumed to exist.

Define Contact Properties

Display the Define Items Used Globally window, as described in “Using the Define Items Used Globally Window” on page 76.

In the Type panel of the Define Items Used Globally window, click the button beside **Contacts**. Click Add at the bottom left of the window. A Contact Properties window for the contact displays for you to enter the appropriate information as follows:

Name	specifies the name for the contact.

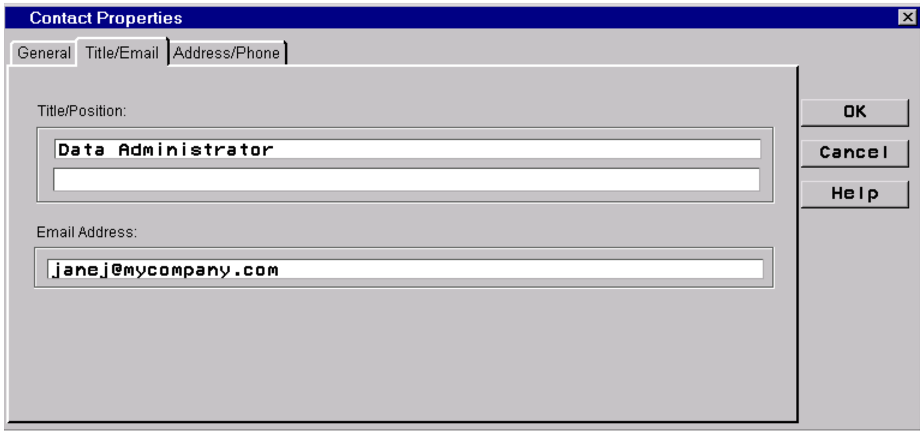


Name: Jane Jones
Description:

The **Name** field contains the name of the contact in our example: **Jane Jones**.

After you have entered a name (and perhaps a description), click the Title/Email tab. The fields on this tab are

Title/ Position	specifies the title of the contact.
Email Address	specifies the e-mail address of the contact.



Title/Position: Data Administrator

Email Address: janej@mycompany.com

The **`Title/Position`** field contains the title of the contact: **`Data Administrator`**.

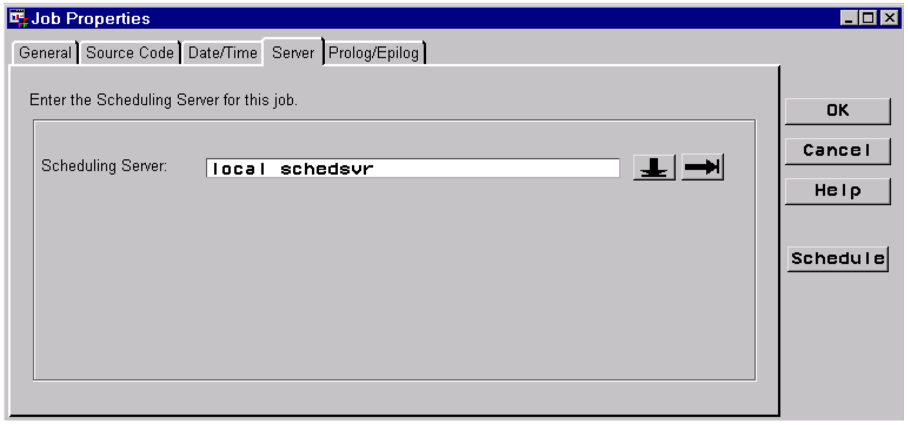
The **`Email Address`** field contains the email address of the contact: **`janej@mycompany.com`**.

After specifying values in these fields, you could click the Address/Phone tab and enter information there.

When you are finished entering information about this contact, click OK. The new contact record is added to the list of contacts in the Define Items Used Globally window.

Scheduling Server Definitions

To have SAS/Warehouse Administrator schedule a Job, you must open the properties window for the Job and enter information on the Date/Time tab and the Server tab. The Server tab requires a scheduling server definition, as shown in the next display:



Enter the Scheduling Server for this job.

Scheduling Server: local schedsvr

A scheduling server definition is a metadata record that specifies

- □ a scheduling server application, such as CRON under UNIX System V, or the AT command under Microsoft Windows or Windows NT
- □ a host definition for the computer where the scheduling server runs
- □ directories where the scheduling server application can send temporary files
- □ the commands used to start SAS on the scheduling server host (to execute the Job)
- □ the default job-tracking option for Jobs using this scheduling server definition.

Scheduling server definitions are required if you want SAS/Warehouse Administrator to generate the code to schedule and track Jobs. At a minimum, in each Warehouse Environment, create a scheduling server definition for each host where scheduled Jobs will run. This typically includes the host where SAS/Warehouse Administrator is installed, as well as one or more remote hosts.

Preparing to Create Scheduling Server Definitions

Before you create a scheduling server definition, make the following preparations:

- □ Create a host definition for the computer where the scheduling server application (such as CRON) runs. For details about host definitions, see “Host Definitions” on page 86.
- □ Identify the kind of scheduling server definition that will support your goals. To schedule and track a Job on a UNIX System V computer, you would typically use a CRON definition. To schedule and track a Job on a Microsoft Windows or Windows NT computer, you would typically use an AT definition. To generate Jobs that can be used with scheduling servers other than AT or CRON, use a Null Scheduler definition. A Null Scheduler definition has other uses as well, as described in “Typical Uses for a Null Scheduler Definition” on page 106.
- □ When scheduling server definitions, specify the physical paths for the directories where the scheduling server application can send temporary files. Create or identify these directories. For example, the working directory for an AT scheduling server definition might be **`c:\job`**.

Example: Creating an AT Scheduling Server Definition

This example describes how to create a scheduling server definition for a local host that is running the Microsoft Windows or Windows NT operating system. Details are provided for a remote host running on one of these operating systems.

Note: The steps for creating a CRON scheduling server definition are very similar. △

Preparing to Create an AT Scheduling Server Definition

In addition to the instructions given in “Preparing to Create Scheduling Server Definitions” on page 100, more preparation might be needed when the AT command is the scheduling server.

The properties window for a scheduling server definition is used to generate a command for a given scheduling server. For example, the Windows NT AT Server Properties window is used to generate an AT command that schedules a SAS/Warehouse Administrator Job.

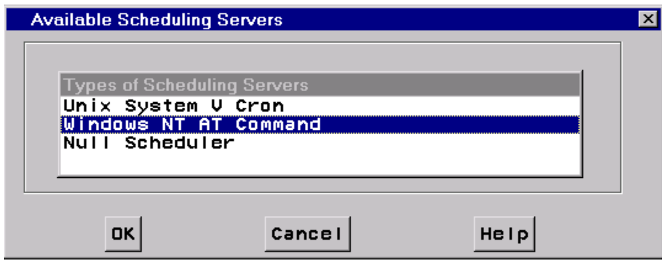
Note: The maximum length for an AT command is 129 characters. △

If the command generated from the metadata in the properties window is too long, you will get an error message that says that the maximum allowable length for an AT command has been exceeded. You will not be able to schedule the Job until this problem has been fixed. To fix this problem, you must shorten the command that is generated from the properties window by shortening pathnames or removing command options.

Define Server Properties

Display the Define Items Used Globally window, as described in “Using the Define Items Used Globally Window” on page 76.

In the Type panel of the Define Items Used Globally window, click the button beside **`Scheduling Servers`**. Click Add at the bottom left of the window. The Available Scheduling Servers window displays.

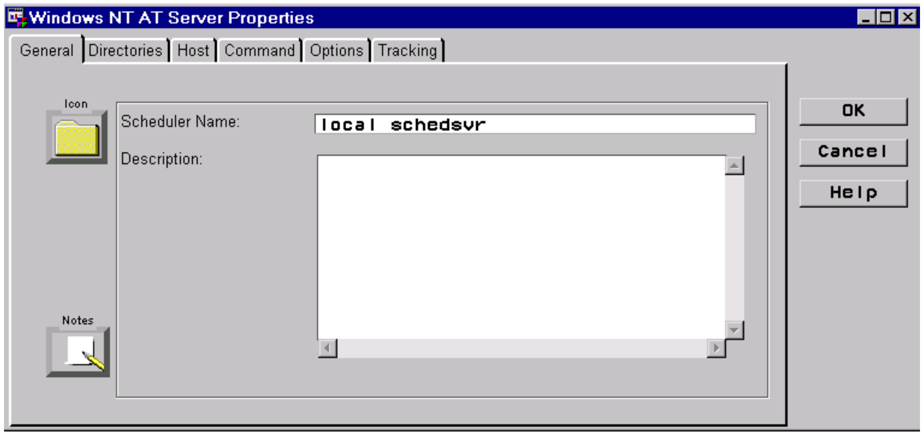

Available Scheduling Servers

Types of Scheduling Servers
Unix System V Cron
Windows NT AT Command
Null Scheduler

For the current example, on the Available Scheduling Servers window, the **`Windows NT AT Command`** is selected. (You would select **`UNIX System V Cron`** to create a scheduling server definition for a host running a System V UNIX, and you would select **`Null Scheduler`** to generate a Job that could be used with a scheduling server other than AT or CRON.)

After you have selected the appropriate scheduling server, click OK. A Windows NT AT Server Properties window for the scheduling server displays for you to enter the appropriate information as follows:

`Scheduler Name`	specifies a display name for the scheduling server definition you are creating, such as **`local schedsvr`**.


Windows NT AT Server Properties

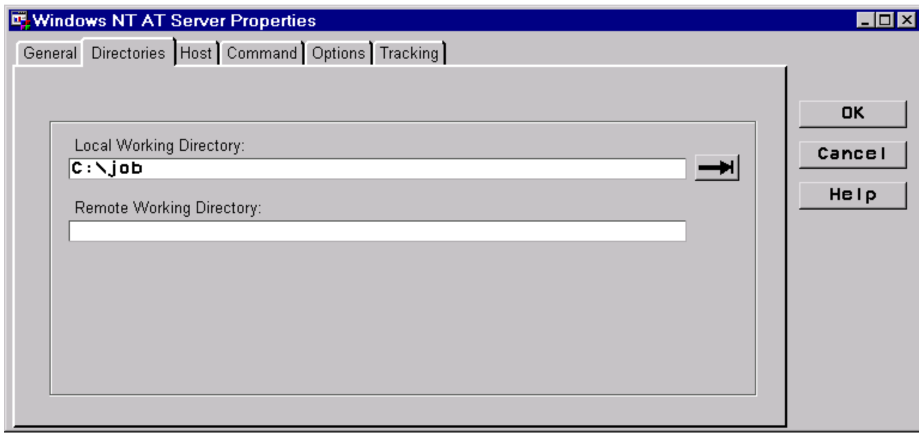
Scheduler Name: local schedsvr

Description:

On the General tab, after you have entered a name (and perhaps a description), click the Directories tab. The fields on this tab are

`Local Working Directory`	specifies a directory that the scheduling server can use to store SYSIN, LOG, LIST, and temporary files that are generated for a Job on the SAS/Warehouse Administrator host.

Note: For the AT command, keep the names of any working directories as short as possible to avoid generating a command that is too long for AT software to accept. △

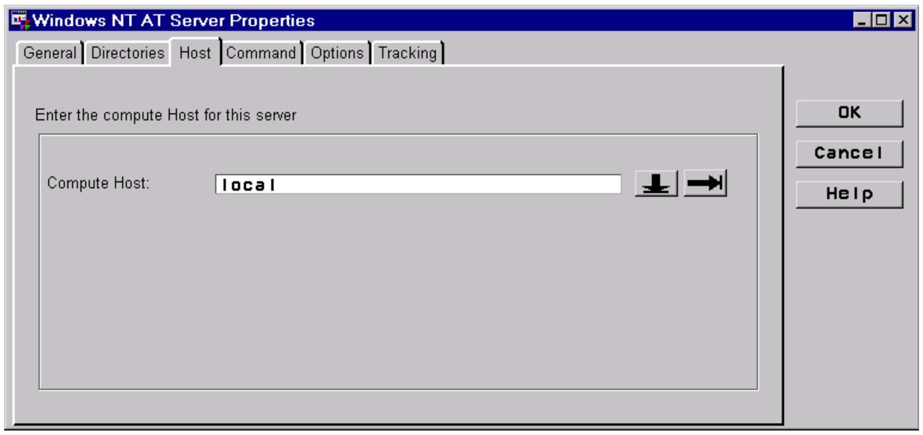


Local Working Directory: C:\job

Remote Working Directory:

The current example is a scheduling server definition for a local host (SAS/Warehouse Administrator host). Accordingly, the Directories tab only specifies a local working directory: **`c:\job`**. If you were creating a scheduling server definition for a remote host, you would specify a working directory on the remote host as well. After specifying the appropriate working directories, click the Host tab. The fields on this tab are

`Compute Host` specifies a definition for the host where the Job will be executed.

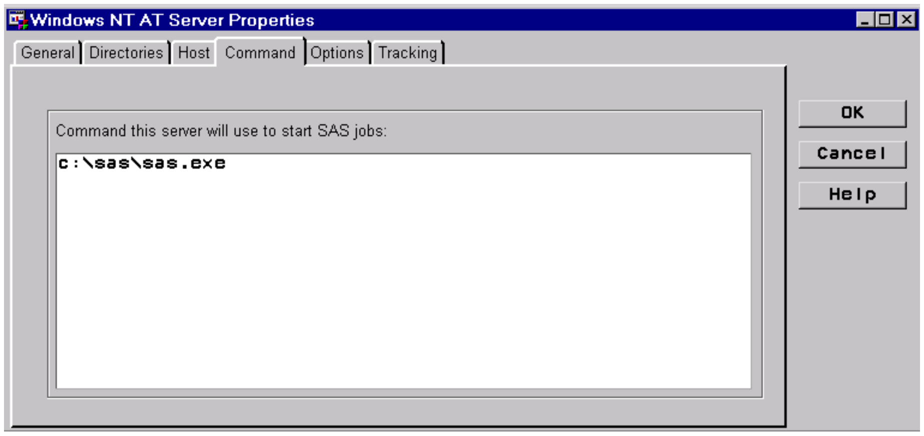


Enter the compute Host for this server

Compute Host: local

The current example is a scheduling server definition for a local host. Accordingly, a local host definition (which happens to be named **`local`**) is specified on the Host tab. If you were creating a scheduling server definition for a remote host, you would specify a remote host definition. After specifying the appropriate host definition, click the Command tab. The fields on this tab are

`Command this server will use to start SAS jobs` specifies the command that the scheduling server application will use to start SAS, in order to execute the Job.



Command this server will use to start SAS jobs:

```
c:\sas\sas.exe
```

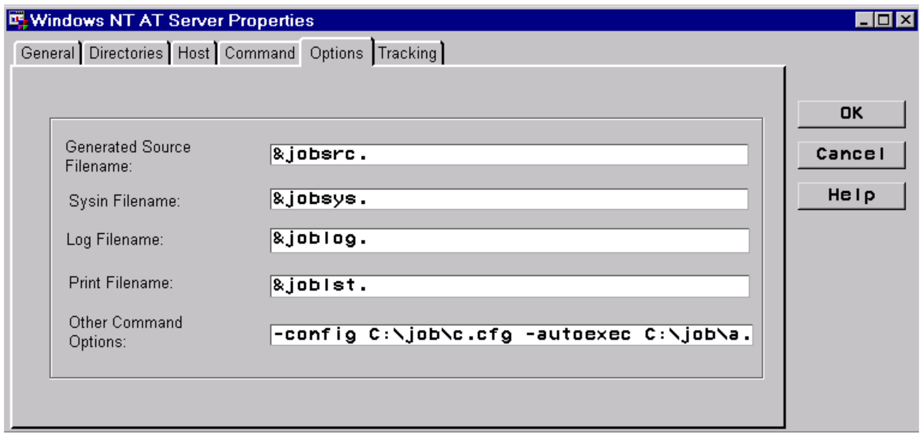
When you first display the Command tab, a default command is provided. You might have to edit the default command to start SAS at your site.

Note: Use the Command tab to specify a SAS start command only. Do not specify a script file on this tab. △

Note: Do not specify a SAS config file, a SAS autoexec file, or any other options on the Command tab. Specify any SAS start command options on the Options tab. △

After specifying the appropriate start command, click the Options tab. The fields on this tab are

Generated Source Filename	specifies the name of the file generated by SAS/Warehouse Administrator for Jobs using this scheduling server definition. By default, a macro is provided in this field that sends the generated file to the local working directory specified on the Directories tab.
Sysin Filename	specifies the default location of the SAS source code to be used during batch mode execution of Jobs using this scheduling server definition. By default, a macro is provided in this field that specifies the **Generated Source Filename** as the **Sysin Filename**.
Log Filename	specifies the filename where you want SAS to write the LOG output. By default, a macro is provided in this field that sends the log to the local working directory specified on the Directories tab.
Print Filename	specifies the filename where you want the output of this Job to be written. By default, a macro is provided in this field that sends any output to the local working directory specified on the Directories tab.
Other Command Options	specify options for the SAS command entered on the Command tab.



Generated Source Filename: &jobsrc.
Sysin Filename: &jobsys.
Log Filename: &joblog.
Print Filename: &joblst.
Other Command Options: -config C:\job\c.cfg -autoexec C:\job\a.

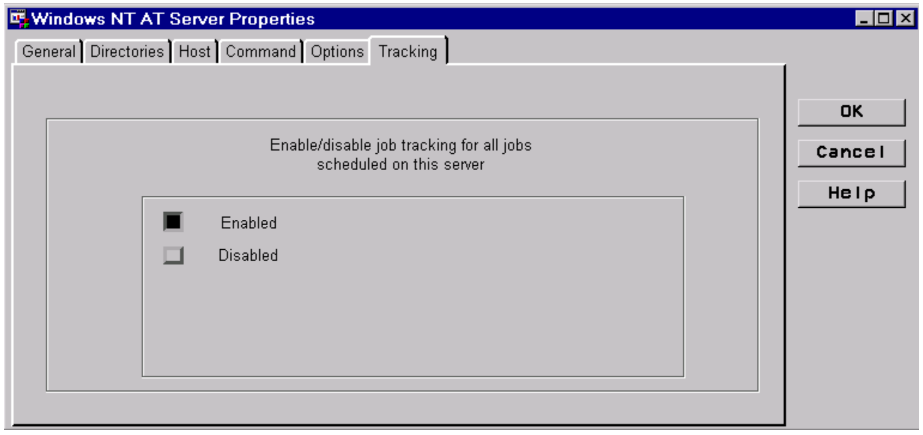
On the Options tab, all of the default macros are specified (**`&jobsrc.`**, **`&jobsys.`**, and so on). One way to shorten the AT command generated from the Windows NT AT Server Properties window is to omit values in the **`Log Filename`** and **`Print Filename`** fields, but that was not required in our example.

A SAS config file and SAS autoexec file with short filenames are specified in the **`Other Command Options`** field. These short filenames are another way to shorten the AT command generated from the properties window.

After specifying the appropriate command options, click the Tracking tab. The fields on this tab are

`Enabled` adds the default job-tracking prolog and epilog to Jobs using this scheduling server definition unless this default is overridden by the tracking attribute for an individual Job.

`Disabled` omits the default job-tracking prolog and epilog from Jobs using this scheduling server definition unless this default is overridden by the tracking attribute for an individual Job.



Enable/disable job tracking for all jobs scheduled on this server

[✓ Enabled] Disabled

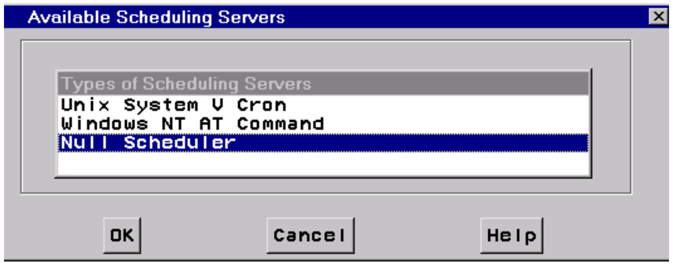
For the current example, accept the default value on the Tracking tab: **`Enabled`**.

After specifying the appropriate metadata on the Tracking tab, you are finished entering metadata for the example scheduling server definition. Click $\boxed{\text{OK}}$. The new server definition is added to the list of server definitions in the Define Items Used Globally window.

After a scheduling server definition has been created, it can be used to schedule a Job, as described in "Example: Scheduling and Tracking a Job with the AT Command" on page 307.

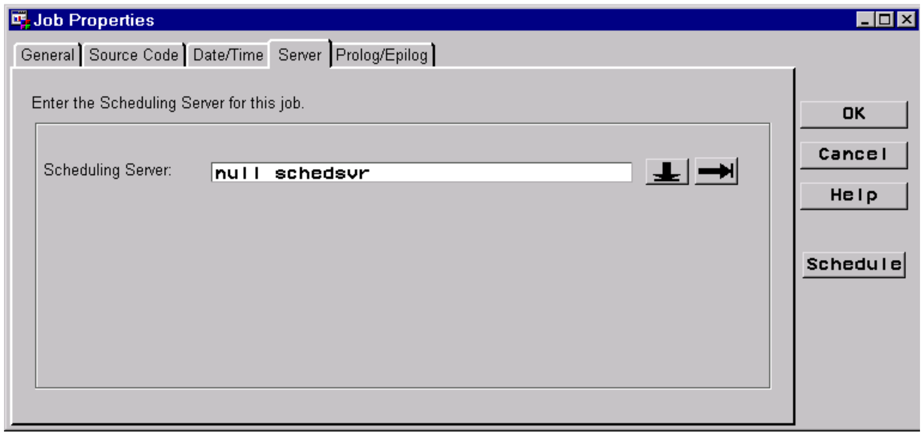
Summary: Creating and Using a Null Scheduling Server Definition

The Available Scheduling Servers window includes a **`Null Scheduler`** option, as shown in the following display:



Use the **`Null Scheduler`** option to create a scheduling server definition that will save a generated SAS Job as a **`.sas`** file in a specified directory, along with a command file to run that Job.

For example, suppose that you wanted to schedule a SAS/Warehouse Administrator Job that would execute on a mainframe computer. You could create a null scheduler definition and use it to schedule the Job. That is, you could define the Job in SAS/ Warehouse Administrator, then schedule it, specifying a null scheduling server definition on the Server tab in the Job Properties window as follows:



Enter the Scheduling Server for this job.

Scheduling Server: null schedsvr

If you click [Schedule] on the Job Properties window, SAS/Warehouse Administrator will

- □ generate the code appropriate for the Job and save it to a **`.sas`** file
- □ generate a SAS start command and write it to a plain text file
- □ write tracking metadata for the Job to the Jobs Information library.

After the command file for the Job has been generated, give it to the mainframe operator who is responsible for scheduling batch jobs. Keep in mind that the mainframe scheduling server would need access to the generated code for the Job, which is specified with the -SYSIN option in the command file.

Typical Uses for a Null Scheduler Definition

You can use a null scheduler definition to support the following scenarios:

- Many sites do not permit users to schedule their own batch jobs, especially jobs that update shared files. If that were the case at your site, you could schedule a SAS/Warehouse Administrator Job with a null scheduler definition. You could then give the resulting command file to the operator who is responsible for scheduling batch jobs. Keep in mind that the scheduling server application would need access to the generated code for the Job, which is specified with the -SYSIN option in the command file.
- If you want to use a scheduling server application other than CRON or AT, create a null scheduler definition and use it to schedule the Job. You could then submit the resulting command file to the scheduling server application. The scheduling sever application would need access to the -SYSIN file specified in the command.
- You could use a null scheduler definition to generate a set of command files with unique Job IDs. These command files could be executed by a program outside of SAS/Warehouse Administrator. For example, you could create a program that executed the Jobs for all of the data stores in a given Subject, Data Warehouse, or Warehouse Environment.

Define Server Properties

Except for selecting **`Null Scheduler`** from the Available Scheduling Servers window, the steps for creating a null scheduler definition are similar to those described for the AT command in "Define Server Properties" on page 100.

Keep in mind that if a local host is specified in the null scheduler definition, the generated SAS Job and its command file will be sent to the directory specified in the **`Local Working Directory`** field on the Directories tab. If a remote host is specified, these files will be sent to the directory specified in the **`Remote Working Directory`** field.

What's Next

After you have created the main global metadata items that you will need in a given Warehouse Environment, you are ready to register the data sources for that Environment.

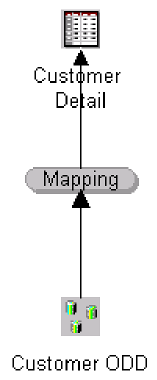
CHAPTER 7 Registering Data Sources

Overview

After you create a Warehouse Environment and the appropriate global metadata, you are ready to create Operational Data Definitions. An Operational Data Definition (ODD) is a metadata record that provides access to data sources. The ODDs are used as inputs to detail data stores or summary data stores in a Warehouse Environment.

For example, the following figure illustrates a Process Flow in which a data store named Customer Detail is fed by the Customer ODD.

Figure 7.1 ODD in a Process Flow



ODDs provide access to source data by

- □ registering the location of a SAS table or view.
- □ registering the location of a DBMS table with the help of a SAS/ACCESS LIBNAME definition.
- □ executing user-written code that extracts information from a data source and then saves the results to a SAS table or view. The location of the extraction table or view is then specified in the ODD.

To create an ODD that registers the location of a SAS table, view, or SAS/ACCESS LIBNAME definition:

1. Open the appropriate Environment in the SAS/Warehouse Administrator Explorer.
2. Add an ODD Group to the Environment. Update the default properties of the group.
3. Add an ODD to the group. Update the default properties of the ODD.
4. Verify that the ODD can successfully generate a LIBNAME statement.

To create an ODD that executes user-written code:

1. Write or generate a SAS routine that creates a table or view and saves it to a SAS library.
2. Open the appropriate Environment in the SAS/Warehouse Administrator Explorer.
3. Add an ODD Group to the Environment. Update the default properties of the group.
4. Add an ODD to the group. Update the default properties of the ODD.
   - □ From the Data Location tab for the ODD, specify the location of the table or view that is created by the routine in step 1.
5. Define a Process Editor Job for the ODD.
   - □ From the Source Code tab for the ODD's Load Step, specify the location of the routine from step 1.
6. Execute the ODD's Job. This executes the routine from step 1.
7. Verify that the ODD provides access to the table or view that is created by the routine from step 1.

Note: The basic steps for maintaining ODDs are described in the online Help. To display the relevant online Help, in the SAS System Help contents, select

Help on SAS Software Products ▶ Using SAS/Warehouse Administrator Software

▶ Getting Data Into Your Warehouse Environment

▶ Maintaining ODD Groups or Maintaining Operational Data Definitions

In addition, you can display Help for most SAS/Warehouse Administrator windows by clicking Help on the window. △

Note: ODDs are the first SAS/Warehouse Administrator data stores that require you to create one or more groups, data stores, Jobs and processes. Accordingly, the descriptions in this chapter are more detailed than the descriptions in later chapters. △

Preparing to Create ODDs

Before you create an ODD Group and its associated ODDs, you must first do some preparation, which is summarized next:

Hierarchy of objects	□ The Data Warehouse Environment that will contain the ODD Group and its ODDs must exist.
Global metadata	□ Create at least some of the appropriate host definitions, SAS library definitions, and other global metadata items that your ODDs require, as described in the Chapter 6, "Maintaining Global Metadata," on page 75.
Input source(s)	□ Determine which input sources(s) to use for each ODD. Identify the host and physical path for each input source.
Output target	□ Determine how many ODDs you need. Your Process Flows might be easier to understand and manage if you have one ODD for each of the main data sources identified in your project plan.

Example: Creating an ODD Group

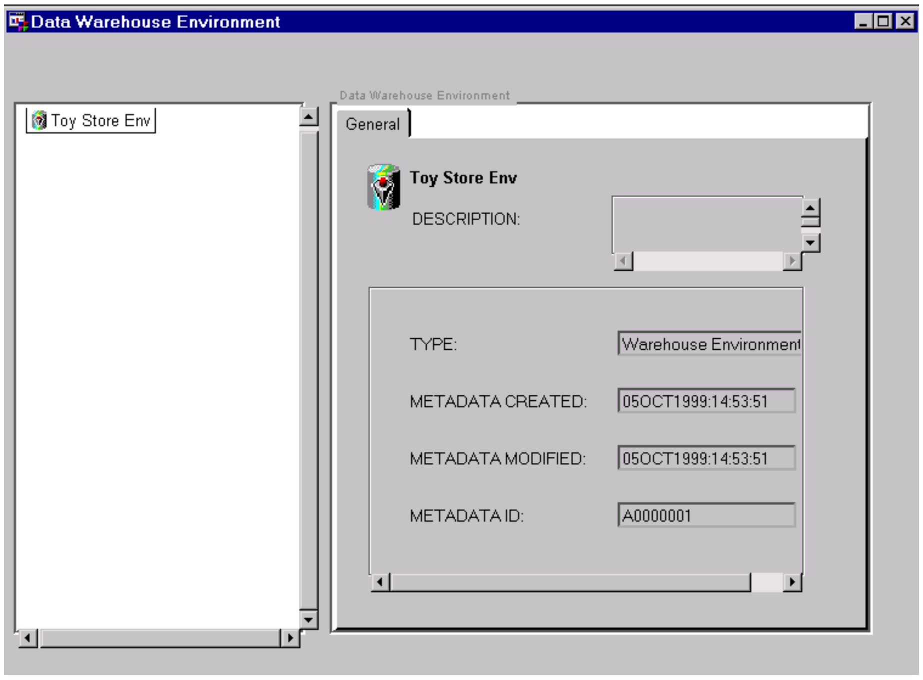
Overview

An ODD Group is a simple grouping element used to organize ODDs and Information Marts. Before you can create an ODD, you must create an ODD Group. This example describes how to add an ODD Group to a Warehouse Environment.

Note: The following explanations describe the metadata and methods used to achieve the desired results; it is assumed that the appropriate Data Warehouse Environment exists. △

Define ODD Group Properties

Open the appropriate Warehouse Environment in the Explorer, as described in "Opening a Warehouse Environment in the Explorer" on page 63. The next display illustrates what a new Environment would look like in the Explorer:

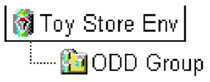


Toy Store Env

DESCRIPTION:

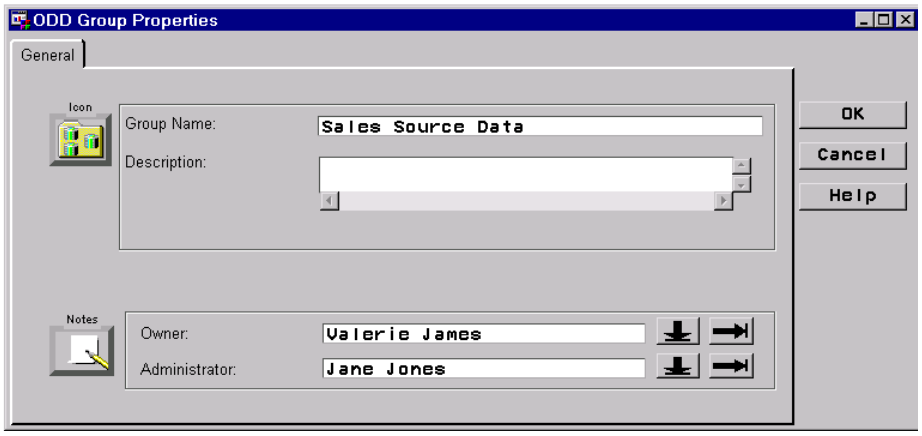
TYPE:	Warehouse Environment
METADATA CREATED:	05OCT1999:14:53:51
METADATA MODIFIED:	05OCT1999:14:53:51
METADATA ID:	A0000001

In the Explorer, position the cursor on the Environment, for example, **`Toy Store Env`**, click the right mouse button, select **`Add Item`**, and then **`Operational Data Definition (ODD) Group`**. In the Explorer window, a new ODD Group is added under the Environment, as follows:



To update the default metadata for the group, position the cursor on its icon, click the right mouse button, and select **`Properties`**. The Data Group Properties window displays for you to enter the appropriate information.

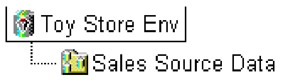
General Tab	specifies the group's name, such as **`Sales Source Data`**, as well as a description, an owner, and an administrator.



Group Name:	Sales Source Data
Description:	
Owner:	Valerie James
Administrator:	Jane Jones

To specify an owner or an administrator, click the down arrow to select a name from a list. If you need to add a new Contact record, click the right arrow to display a properties window. For details about that window, click its Help button.

After specifying the appropriate values, click OK. You will be returned to the Explorer. The new ODD Group will appear under the Environment, as follows:



Example: Creating an ODD That Registers the Location of a Data Source

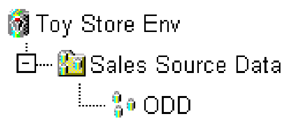
Overview

This example describes how to create an ODD that registers the location of a SAS table or view, or one that registers the location of a DBMS table with the help of a SAS/ ACCESS LIBNAME definition. Use this approach when you want to bring source data into a Warehouse Environment without transforming it at the ODD level. (You can transform this data later in a Process Flow—between an ODD and a data store that it feeds, for example.)

Note: The following explanations describe the metadata and methods to achieve the desired results; it is assumed that the appropriate ODD Group exists. △

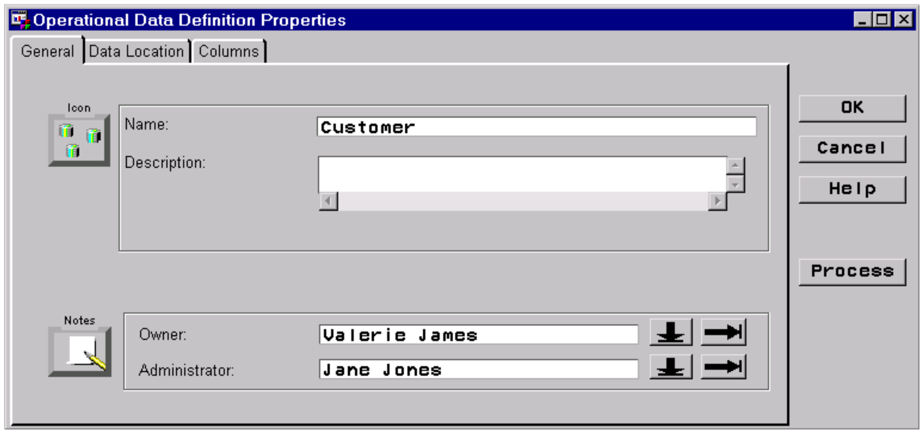
Define ODD Properties

In the SAS/Warehouse Administrator Explorer, position the cursor on the parent ODD Group, for example, **`Sales Source Data`**, click the right mouse button, select **`Add Item`**, and then **`ODD`**. In the Explorer window, a new ODD is added under the ODD Group as follows:



To update the default metadata for the ODD, position the cursor on its icon, click the right mouse button, and select **`Properties`**. The Operational Data Definition Properties window displays for you to enter the appropriate information.

General Tab	specifies the ODD's name, such as **`Customer`**, as well as a description, an owner, and an administrator.



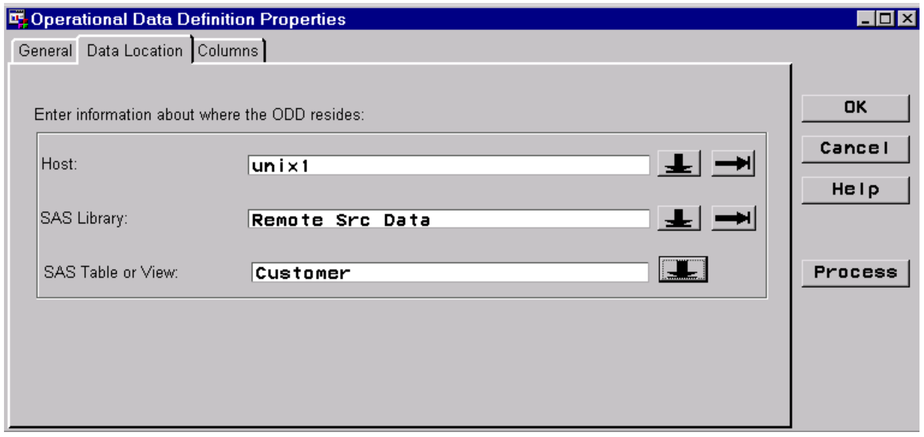
Name: Customer
Description:
Owner: Valerie James
Administrator: Jane Jones

To specify an owner or an administrator, click the down arrow to select a name from a list. If you need to add a new Contact record, click the right arrow to display a properties window. For details about that window, click its Help button.

The next task is to specify the location of the data source for the ODD. Two examples of metadata for the Data Location tab are shown next.

Example 1: Data Location tab for a remote SAS table

If you want to register a remote SAS table, specify the definition for the host where the target library resides (such as **`unix1`**); specify the target SAS library (such as **`Remote Src Data`**); and specify the SAS table or view within that library (such as **`Customer`**). The next display shows an example of this metadata:



Enter information about where the ODD resides:
Host: unix1
SAS Library: Remote Src Data
SAS Table or View: Customer

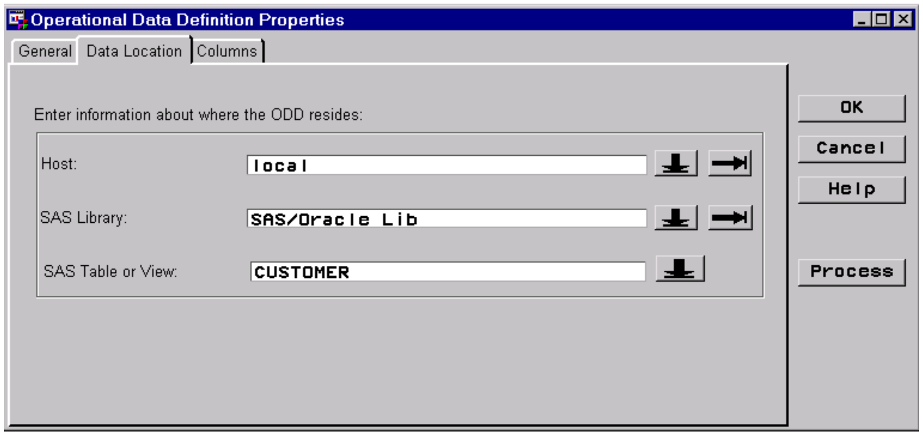
To specify a host or SAS library, click the down arrow to select a name from a list. If you need to add a new host or library, click the right arrow to display a properties window. For details about that window, click its Help button.

To specify a SAS table or view within the library, click the down arrow beside the **`SAS Table or View`** field to select a name from a list.

Note: If you can select a SAS table by clicking the down arrow beside the **`SAS Table or View`** field, then SAS was able to assign the library that is specified in the **`SAS Library`** field. If you are unable to access a table, read the SAS log to identify the problem. △

Example 2: Data Location tab for a local SAS/ACCESS LIBNAME

If you want to register a local SAS/ACCESS LIBNAME for a remote DBMS table, specify the definition for the host where the SAS/ ACCESS LIBNAME statement will be executed (such as **`local`**); specify the SAS/ACCESS LIBNAME (such as **`SAS/Oracle Lib`**); and specify the DBMS table within the DBMS (such as **`Customer`**). The next display shows an example of this metadata:



Enter information about where the ODD resides:

Host: local

SAS Library: SAS/Oracle Lib

SAS Table or View: CUSTOMER

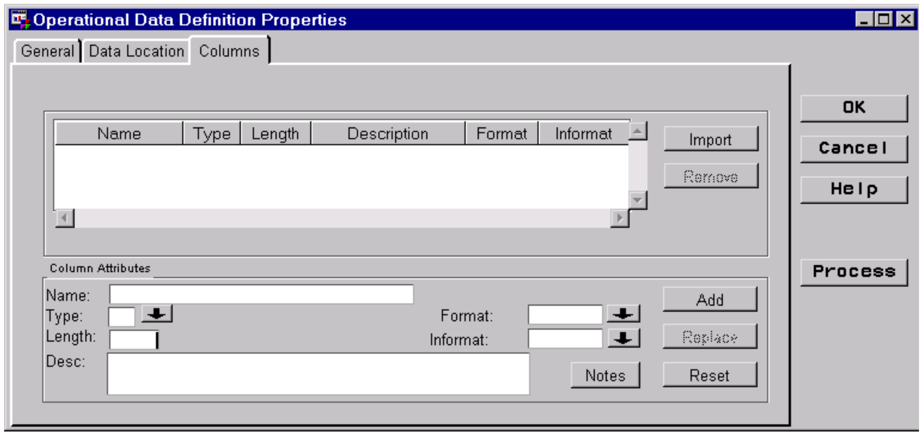
To specify a host or a SAS/ACCESS LIBNAME, click the down arrow to select a name from a list. If you need to add a new host or SAS/ACCESS LIBNAME, click the right arrow to display a properties window. For details about that window, click its Help button.

To specify a DBMS table that is accessed via the local SAS/ ACCESS LIBNAME, click the down arrow beside the **`SAS Table or View`** field to select a name from a list.

Note: If you can select a DBMS table by clicking the down arrow beside the **`SAS Table or View`** field, then SAS was able to assign the SAS/ACCESS LIBNAME that is specified in the **`SAS Library`** field. If you are unable to access a table, read the SAS log to identify the problem. △

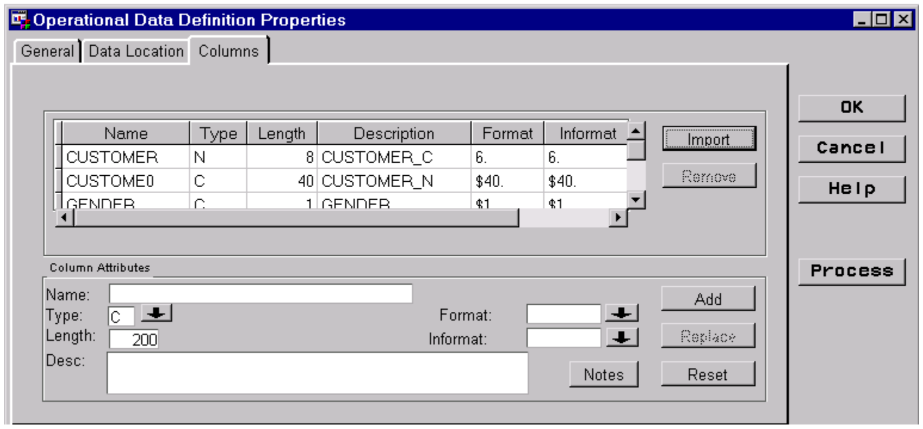
The next task is to specify column metadata for the ODD.

From the Columns tab, you can specify the column metadata to be included in the ODD. No column metadata has been specified yet, as shown next:



The column metadata for a data store must accurately specify the columns that you want to "map" to other data stores in a Process Flow, such as the one shown in Figure 7.1 on page 108. In the case of an ODD that simply registers the location of an existing data source, you will use the "Import from Supplied Data Location" method to import the metadata for all columns in the data source.

To import columns from the data source, select Import, then select **Supplied Data Location**. All of the columns from the data source specified on the Data Location tab are imported, as follows:



Name	Type	Length	Description	Format	Informat
CUSTOMER	N	8	CUSTOMER_C	6.	6.
CUSTOME0	C	40	CUSTOMER_N	$40.	$40.
GENDER	C	1	GENDER	$1	$1

Name:
Type: C
Length: 200
Desc:
Format:
Informat:

After you specify the column metadata, you are finished creating the ODD. Click OK to save its metadata record and return to the Explorer.

No Jobs for ODDs That Only Register Locations

A Job is a metadata record that specifies the processes that create one or more data stores (output tables). You create Jobs in the Process Editor window. Most data stores in SAS/Warehouse Administrator require Jobs that execute code. However, if an ODD simply registers the location of a data source and it does not execute any code, then no Job is needed to create the ODD.

Test the ODD

To verify that an ODD can successfully generate a LIBNAME statement (as it will have to do in a Process Flow), position the cursor on the ODD's icon in the Explorer, click the right mouse button, and select **`Assign Libref`**. Check the SAS log or the Active Libraries panel of the SAS Explorer window to verify that the SAS library associated with this ODD was assigned.

After you have tested your ODD, you can use it as an input to another data store. See also the maintenance note in "Keeping ODD Column Metadata Current" on page 130.

Example: Creating an ODD with a User-Written Load Step

Overview

This example describes how to create an ODD that executes user-written code that extracts information from a data source, and then saves the results to a SAS table or view. The location of the extraction table or view is then specified in the ODD. Use this approach when you want to transform source data before making it available in a Warehouse Environment.

Note: The following explanations describe the metadata and methods to achieve the desired results; it is assumed that the appropriate ODD Group exists. △

Preparing to Create an ODD with a User-Written Load Step

In addition to the preparation described in "Preparing to Create ODDs" on page 109, you must do the following preparation for ODDs with user-written Load Steps:

Input Sources	□ Analyze the columns in the input data and determine which columns you want to include in the ODD.
Output Targets	□ Determine on what host and in what SAS library you will store your user-written Load Step routine and the table or view that is output from this routine.
Processes	□ Determine how you will create the SAS code for the Load Step.
	□ Determine if the ODD's Process Editor Job will require processes in addition to the Load Step, such as a User Exit process, a Data Transfer process, or a Record Selector process.

Methods for Creating User-Written Load Step Routines

Load Steps create tables or views. SAS provides a number of ways to create tables and views, including

- □ the DATA step
- □ the SQL procedure
- □ the Query window
- □ the External File Interface (EFI)
- □ SAS/ASSIST software
- □ SAS/TOOLKIT software.

The example in this section describes how to use the Query window to generate the majority of a Load Step routine. For more information about user-written source code, see the online Help. To display the relevant online Help, in the SAS System Help contents, select

Help on SAS Software Products ▶ Using SAS/Warehouse Administrator Software

▶ Overview ▶ Overview of SAS/Warehouse Administrator

▶ user-written source code

to display the topic User-Written Source Code.

Create a Load Step Routine (Query Window Method)

This section summarizes how to use the Query window to generate the majority of a Load Step routine. The Query window is an interactive interface that enables you to build, save, and run queries without being an expert with SQL (Structured Query Language) or with the SAS SQL procedure. The query that you build in the Query window is passed to the SAS SQL procedure for processing when you run the query.

The code generated by the Query window can, with minor changes, be used to create a SAS table or view. If you know a little about PROC SQL syntax, this method might be easier than writing your own code to create a Load Step in SAS/Warehouse Administrator.

This section summarizes how to

- use the Query window to generate a query against a data source in an input library
- save the query in a code library
- edit the query so that it creates a table or view and saves it to an output library.

Note: The input library, output library, and the code library must have library definitions in the Warehouse Environment that will contain the ODD. △

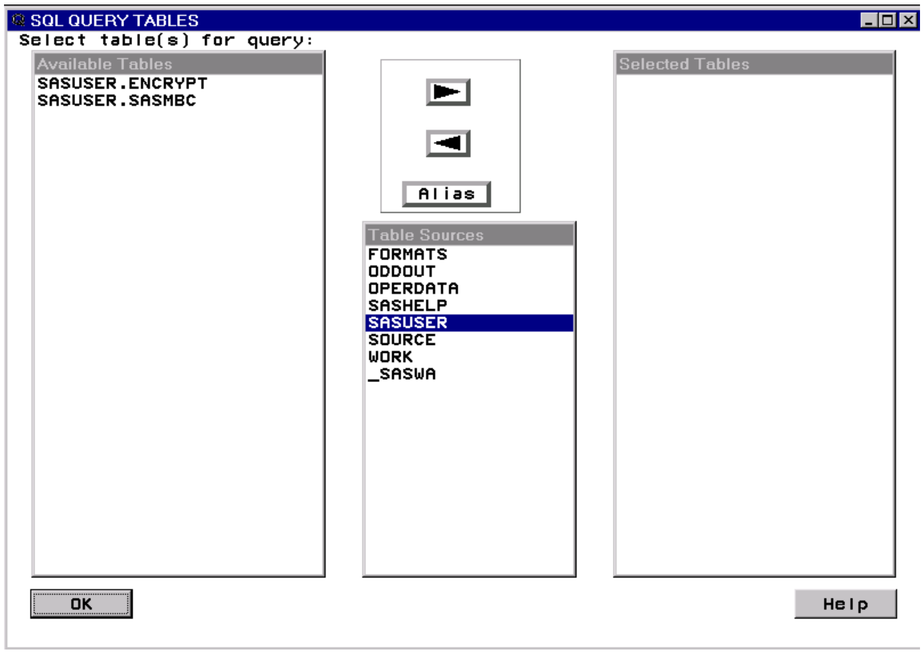
For example, assume that the following SAS libraries have library definitions in the example Toy Store Environment:

- OPERDATA (Operational Data library), where the input table for the query routine is stored
- SOURCE (Source Code library), where the query routine will be saved
- ODDOUT (ODD Output library), where the edited query will write its output.

Use the Query Window to Generate a Query

1. Assign the input library (such as **`OPERDATA`**) and the code library (such as **`SOURCE`**) in the current SAS session.
2. From the SAS window, select **`Tools`**, then **`Query`** from the menu.

   The SQL Query Tables window displays.

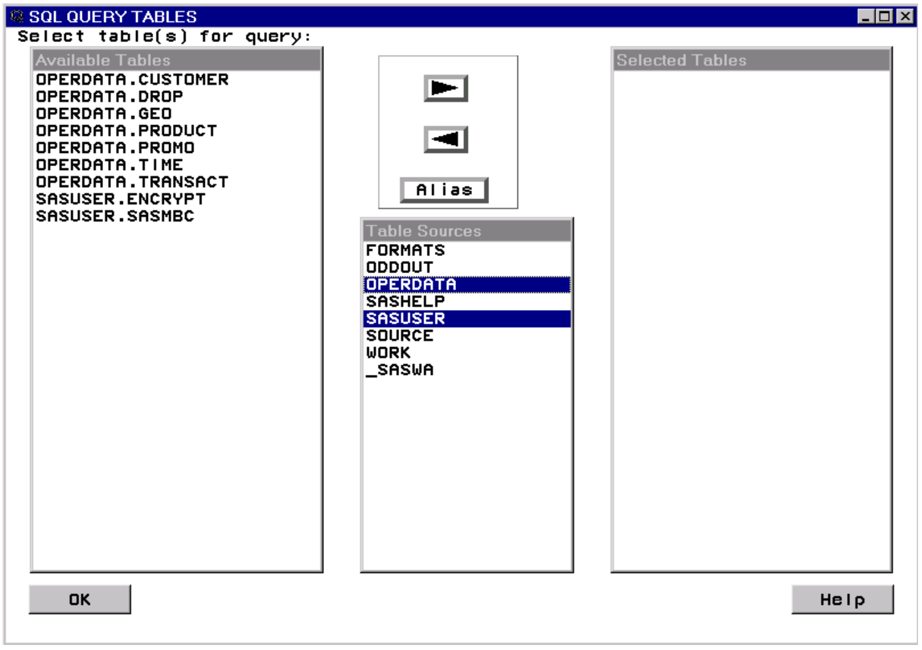


```
SASUSER.ENCRYPT
SASUSER.SASMBC
```

```
FORMATS
ODDOUT
OPERDATA
SASHELP
SASUSER
SOURCE
WORK
_SASWA
```

3 From the SQL Query Tables window, in the Table Sources panel, click the input library where the data source resides (such as **`OPERDATA`**).

The tables in the selected library are listed in the Available Tables panel.

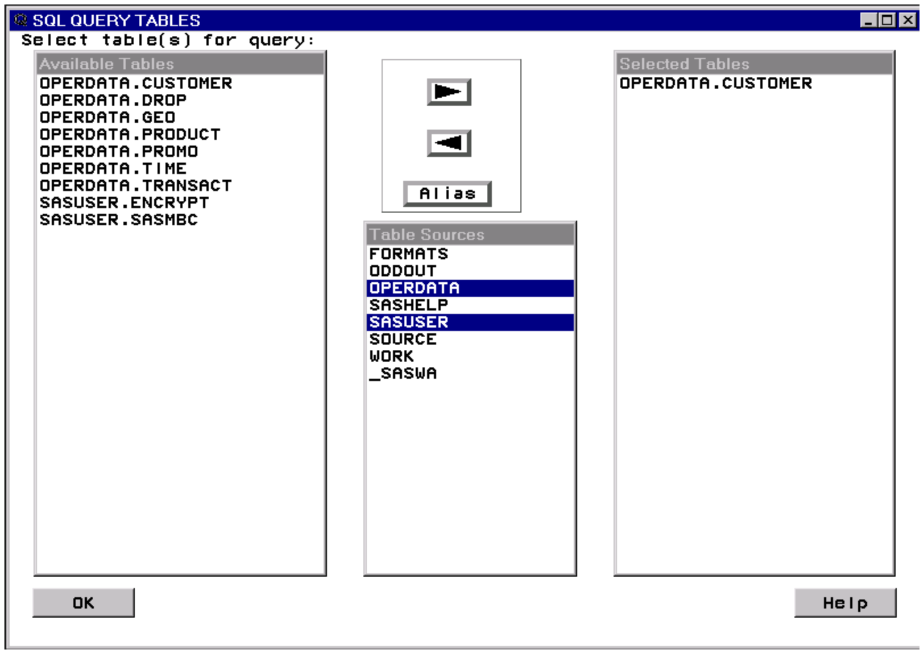


```
OPERDATA.CUSTOMER
OPERDATA.DROP
OPERDATA.GEO
OPERDATA.PRODUCT
OPERDATA.PROMO
OPERDATA.TIME
OPERDATA.TRANSACT
SASUSER.ENCRYPT
SASUSER.SASMBC
```

```
FORMATS
ODDOUT
OPERDATA
SASHELP
SASUSER
SOURCE
WORK
_SASWA
```

4 In the Available Tables panel, click the table that contains the data for the ODD, and then click the right arrow.

The table is moved to the Selected Tables panel.



SQL QUERY TABLES

Select table(s) for query:

Available Tables

```
OPERDATA.CUSTOMER
OPERDATA.DROP
OPERDATA.GEO
OPERDATA.PRODUCT
OPERDATA.PROMO
OPERDATA.TIME
OPERDATA.TRANSACT
SASUSER.ENCRYPT
SASUSER.SASMBC
```

Table Sources

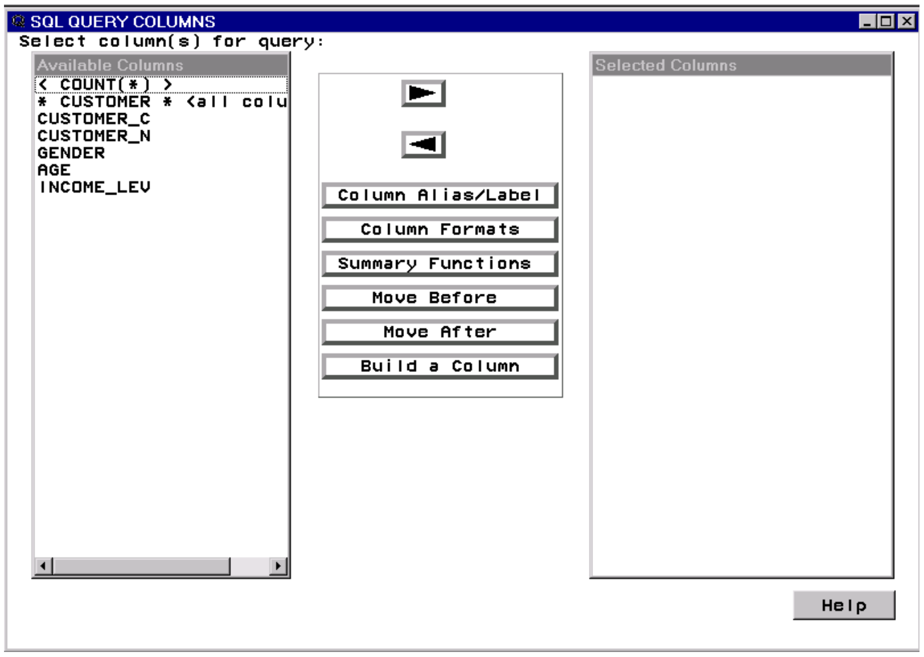
```
FORMATS
ODDOUT
OPERDATA
SASHELP
SASUSER
SOURCE
WORK
_SASWA
```

Selected Tables

```
OPERDATA.CUSTOMER
```

5 In the Selected Tables window, click the table, and then click OK.

The SQL Query Columns table displays.



SQL QUERY COLUMNS

Select column(s) for query:

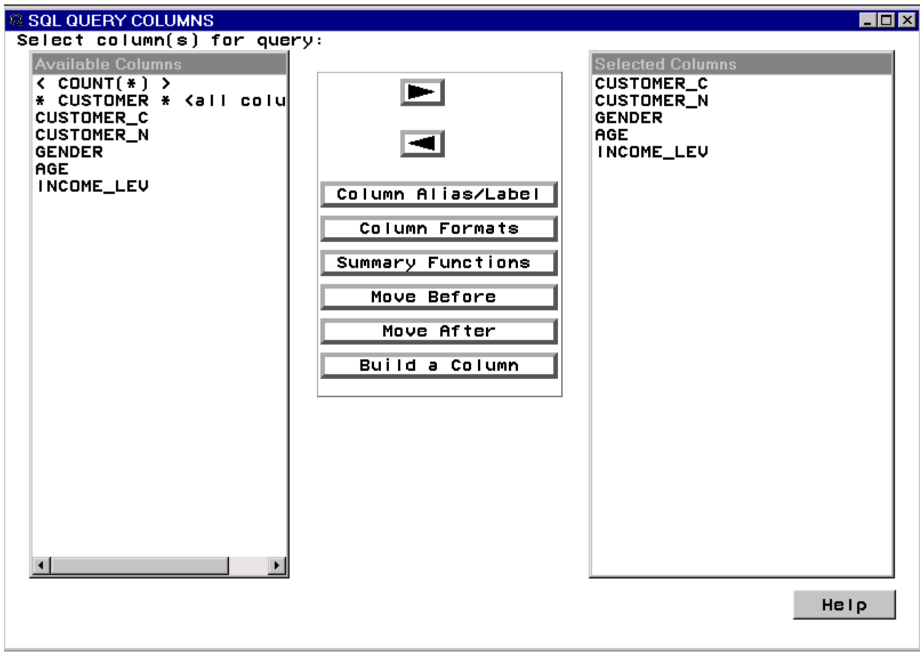
Available Columns

```
< COUNT(*) >
* CUSTOMER * <all colu
CUSTOMER_C
CUSTOMER_N
GENDER
AGE
INCOME_LEV
```

Selected Columns

6 In the SQL Query Columns window, click on the columns you want to appear in the ODD, or select **`all columns`** if you want all columns from the source to appear in the ODD. Then click the right arrow.

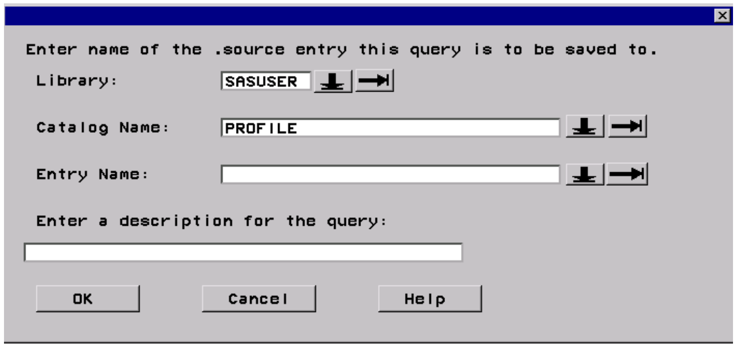
The selected columns move to the Selected Columns window.



7 From the SAS menu, select

File ▶ Save Query ▶ Save Query as a SOURCE entry

A default SOURCE entry specification window appears.


Enter name of the .source entry this query is to be saved to.

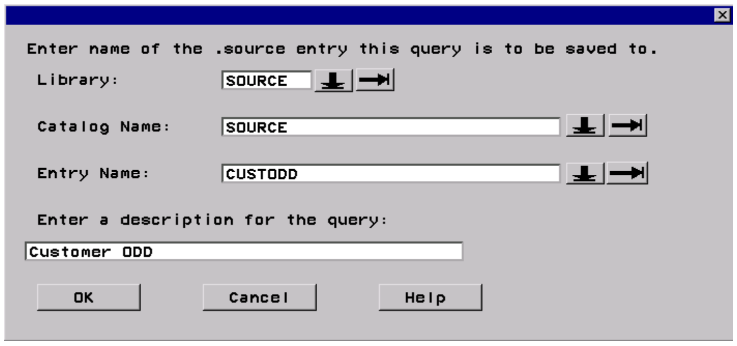
Library: SASUSER

Catalog Name: PROFILE

Entry Name:

Enter a description for the query:

8 Update the SOURCE entry specification window so that it will save your query in the appropriate code library (such as **`SOURCE`**), in the appropriate catalog (which could also be named **`SOURCE`**), and in the appropriate catalog entry (such as **`CUSTODD`**), as follows:


Enter name of the .source entry this query is to be saved to.

Library: SOURCE

Catalog Name: SOURCE

Entry Name: CUSTODD

Enter a description for the query:

Customer ODD

9 Click OK.

Your query is saved. The SQL Query Columns window displays.

10 Exit or minimize the SQL Query Columns window.

You have just generated a query that selects columns from a data source. You must now modify the query so that it will store the query results in another table—the table the ODD will point to.

Edit the Query So That It Creates a Table

The next task is to edit the query generated from the Query window so that it creates a table or view and saves it to the output library. You might only need to add a single line of code—an SQL CREATE TABLE statement or a CREATE VIEW statement—to the code that was generated by the Query window.

Note: The steps below assume that the code library is in SAS Version 8 format. If your code library is not in Version 8 format, the steps for editing a source code entry will be somewhat different than the following steps. △

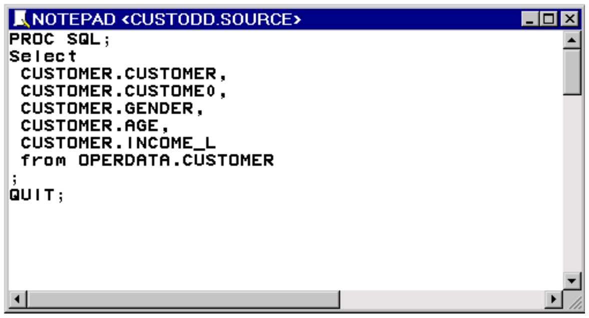
To edit a SAS Version 8 source code entry:

1 In SAS Version 8, if you have not done so already, display the tree view of the SAS Explorer by typing **`explorer`** on the Command line and pressing Return.

2 Assign the input library (such as OPERDATA), the code library (such as SOURCE), and the output library (such as ODDOUT) in the current SAS session.

3 In the Explorer, expand the code library until you can see the entry for the query that you saved from the Query window. Double-click that entry.

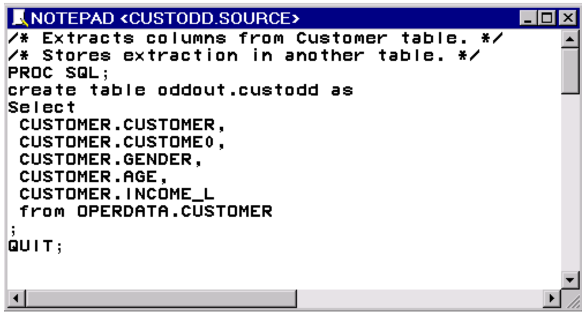
The query routine opens in a SAS Notepad window.



```
PROC SQL;
Select
 CUSTOMER.CUSTOMER,
 CUSTOMER.CUSTOME0,
 CUSTOMER.GENDER,
 CUSTOMER.AGE,
 CUSTOMER.INCOME_L
 from OPERDATA.CUSTOMER
;
QUIT;
```

4 Modify the query so that it will store the query results in a table. Above the first line (**`PROC SQL`**), you might want to add comments that will describe the purpose of this source code. Between **`PROC SQL`** and **`Select`**, add an SQL CREATE TABLE or CREATE VIEW statement that saves its output to an output library (such as ODDOUT).

For example,



```
/* Extracts columns from Customer table. */
/* Stores extraction in another table. */
PROC SQL;
create table oddout.custodd as
Select
 CUSTOMER.CUSTOMER,
 CUSTOMER.CUSTOME0,
 CUSTOMER.GENDER,
 CUSTOMER.AGE,
 CUSTOMER.INCOME_L
 from OPERDATA.CUSTOMER
;
QUIT;
```

5 When you are finished making the modifications, click the **X** in the upper right-hand corner of the SAS Notepad window to save your changes.

You have just modified a query so that it will store the query results in another table—the table that the ODD will point to. The next step is to test the query in the SAS Editor window.

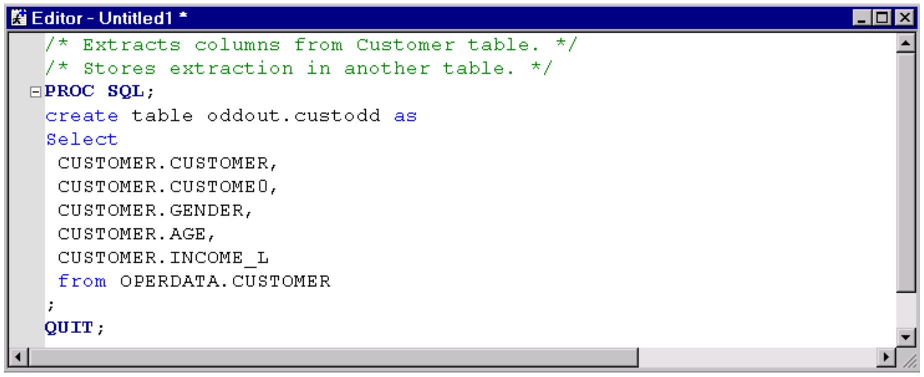
Test the Edited Query in the SAS Editor

After you have edited and saved the query, submit it in the SAS Editor window in order to verify that the routine will in fact create a table. Also, if the table already exists, you can simply import its column metadata in the ODD that you will create later.

1 Assign the input library (such as OPERDATA), the code library (such as SOURCE), and the output library (such as ODDOUT) in the current SAS session.

2 Open the query routine in the SAS Notepad window, as described in "Edit the Query So That It Creates a Table" on page 120.

3 Copy the routine from the SAS Notepad window to the SAS Editor window.



```
/* Extracts columns from Customer table. */
/* Stores extraction in another table. */
PROC SQL;
create table oddout.custodd as
Select
 CUSTOMER.CUSTOMER,
 CUSTOMER.CUSTOME0,
 CUSTOMER.GENDER,
 CUSTOMER.AGE,
 CUSTOMER.INCOME_L
 from OPERDATA.CUSTOMER
;
QUIT;
```

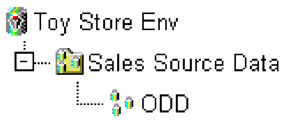
4 Execute the routine. Check the SAS log to verify that the routine creates the table in the output library.

You have just verified that the modified query will create a table in the output library. Later, this routine will be specified in the metadata for the ODD's Load Step. Before you do that, however, you must first define the properties of the ODD.

Define ODD Properties

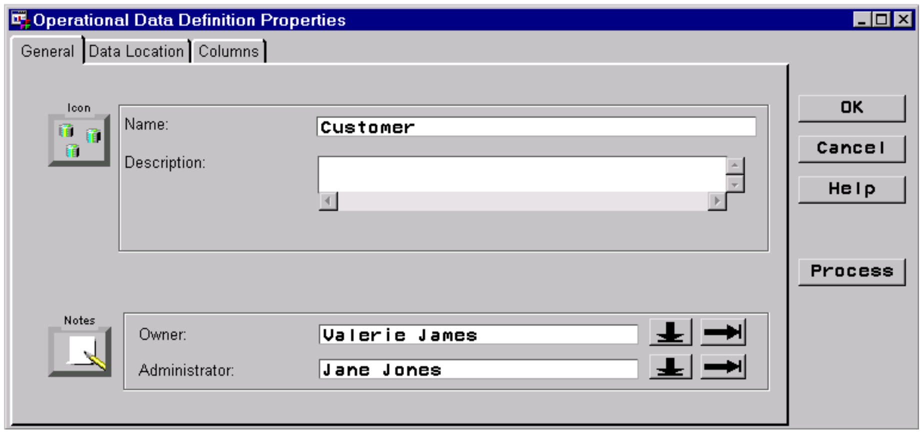
After you have created a routine that extracts information from a data source, and then saves the results to a SAS table or view, you can specify the location of the extraction table or view in an ODD.

In the SAS/Warehouse Administrator Explorer, position the cursor on the parent ODD Group, for example, **Sales Source Data**, press the right mouse button, select **Add Item**, then **ODD**. In the Explorer window, a new ODD is added under the ODD Group.



To update the default metadata for the ODD, position the cursor on its icon, click the right mouse button, and select **Properties**. The Operational Data Definition Properties window displays for you to enter the appropriate information.

General Tab — specifies the ODD's name, such as **Customer**, as well as a description, an owner, and an administrator.

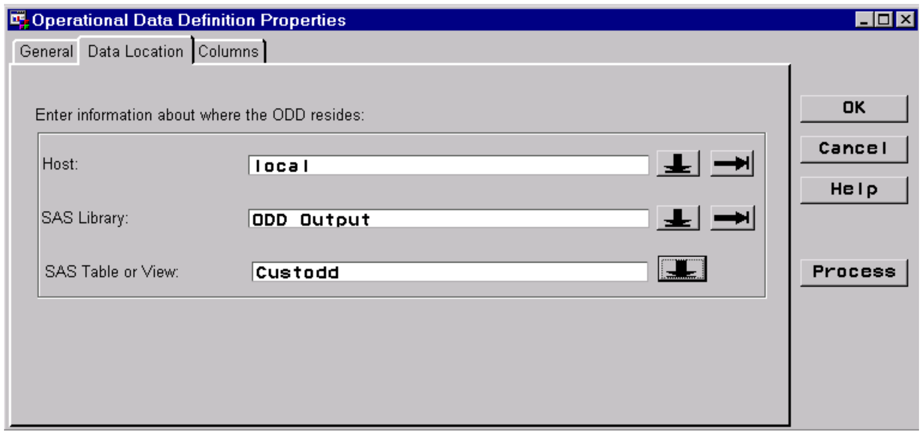


Name: Customer
Description:
Owner: Valerie James
Administrator: Jane Jones

To specify an owner or an administrator, click the down arrow to select a name from a list. If you need to add a new Contact record, click the right arrow to display a properties window. For details about that window, click its Help button.

The next task is to specify the location of the data source for this ODD.

Data Location Tab — specifies the definition for the host where the target library resides (such as **local**); specifies the target SAS library (such as **ODD Output**, the display name for the ODDOUT library); specifies the SAS table or view within that library (such as **Customer**). The next display shows an example of this metadata:



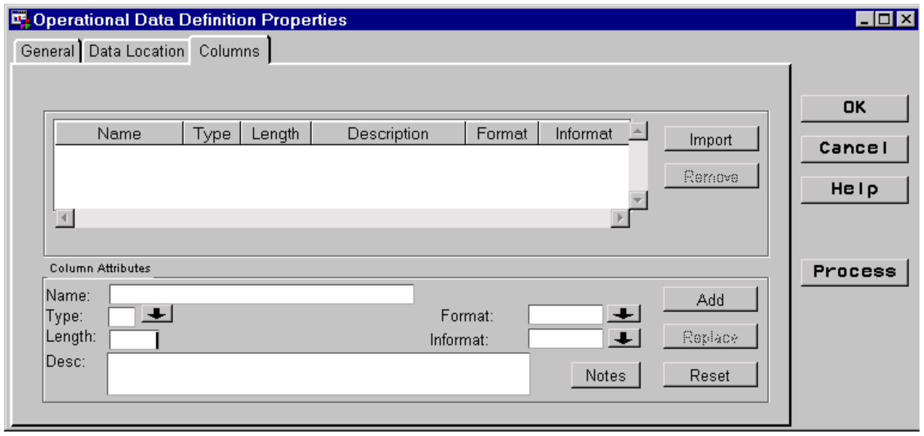
Enter information about where the ODD resides:
Host: local
SAS Library: ODD Output
SAS Table or View: Custodd

To specify a host or SAS library, click the down arrow to select a name from a list. If you need to add a new host or library, click the right arrow to display a properties window. For details about that window, click its Help button.

To specify a SAS table or view within the library, click the down arrow beside the **`SAS Table or View`** field to select a name from a list.

Note: If you can select a SAS table by clicking the down arrow beside the **`SAS Table or View`** field, then SAS was able to assign the library that is specified in the **`SAS Library`** field. If you are unable to access a table, read the SAS log to identify the problem. △

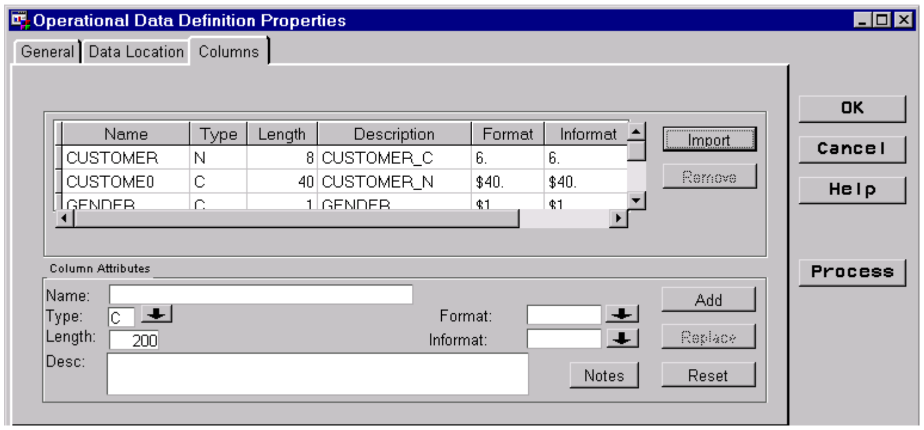
Columns Tab specifies the column metadata to be included in the ODD. No column metadata has been specified yet.



The column metadata for a data store must accurately specify the columns that you want to "map" to other data stores in a Process Flow, such as the one shown in Figure 7.1 on page 108. In the case of an ODD that executes user-written code that extracts information from a data source, and then saves the results to an output table or view, the metadata on this tab should match the columns in the output table or view.

In the current example, the edited query routine has been executed at least once, as described in “Test the Edited Query in the SAS Editor” on page 121. Accordingly, the ODD’s output table already exists. This means that you can use the "Import from Supplied Data Location" method to import the metadata for all columns in the output table.

To import columns from an output table, click Import, and then select **`Supplied Data Location`**. All of the columns from the output table specified on the Data Location tab are imported.



Name	Type	Length	Description	Format	Informat
CUSTOMER	N	8	CUSTOMER_C	6.	6.
CUSTOME0	C	40	CUSTOMER_N	$40.	$40.
GENDER	C	1	GENDER	$1	$1

Name:
Type: C
Length: 200
Desc:
Format:
Informat:

After you specify the column metadata, you have finished entering properties for this ODD. Click OK to save its metadata record and return to the Explorer.

You have just defined the properties of the ODD. You have specified the location of the output table and have specified metadata for the columns in the output table. The next step is to define a Process Editor Job for the ODD.

Define Process Editor Job

In the current example, we have created a routine that extracts information from a data source, and then saves the results to a SAS table or view. We have defined an ODD that specifies the location of the SAS table or view. The next step is to create a Process Editor Job that includes

- a Process Flow that specifies how data moves from the original data source to the ODD
- an ODD Load Step that points to the routine that extracts information from a data source, and then saves the results to a SAS table or view.

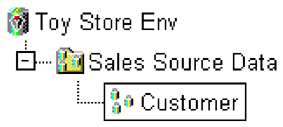
As explained in the “Maintaining Jobs” chapter, data stores with user-written Load Steps—such as the ODD in our current example—do not need Process Flows in the Process Editor. However, creating a Process Flow for the example ODD has two advantages:

- The flow of information from the input source to the ODD will be documented within SAS/Warehouse Administrator.
- SAS/Warehouse Administrator can generate LIBNAME statements for the input source(s) specified in a Process Flow. In our example, this means that you will not have to assign the following SAS libraries manually, which have library definitions in the Toy Store Warehouse Environment:
  - OPERDATA (Operational Data library), where the input table for the Load Step routine is stored
  - SOURCE (Source Code library), where the Load Step routine is stored
  - ODDOUT (ODD Output library), where the Load Step routine will write its output.

  In the current example, these libraries will be specified in the data stores and processes in a Process Flow. When the Process Flow is executed, SAS/Warehouse Administrator assigns the libraries.

Create a Job for the Customer ODD

1 Display the Customer ODD in the SAS/Warehouse Administrator Explorer.

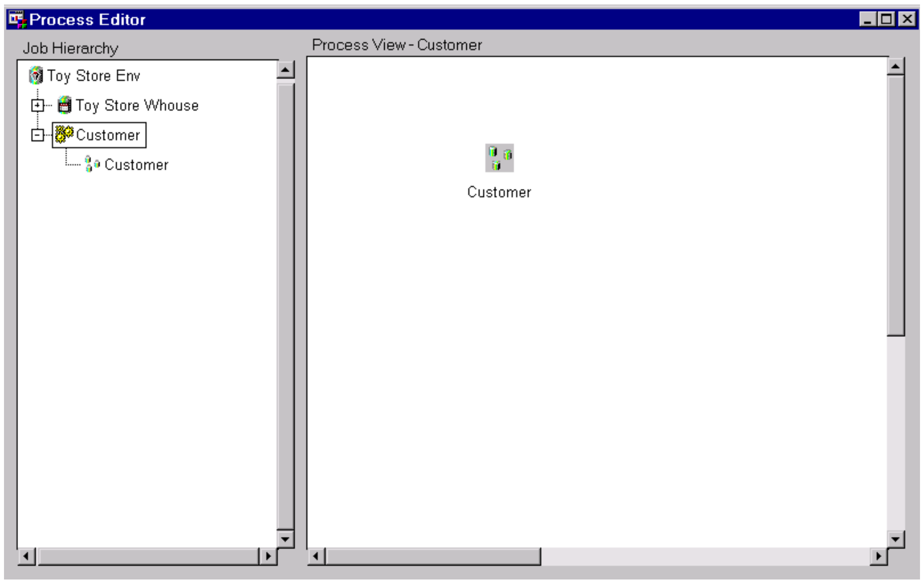


2 Position the cursor on the Customer ODD, click the right mouse button and select **`Process`**.

You will be asked if you want to create a Job for the ODD.

3 Select **`Yes`**.

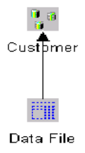
A default Job will be created in the Process Editor, as follows:



In this display, the Job is represented by the icon with the rectangle around it in the left panel. The next task is to specify the input to the Customer ODD. Assume that the original data source in our example is the SAS table OPERDATA.CUSTOMER.

4 In the Process View of the Process Editor (right panel), position the cursor on the Customer ODD, click the right mouse button and select **`Add`**, then **`New Data File`**. (We selected **`New Data File`** because this input has not yet been specified for any other data store in the current Environment, and a Data File is the appropriate type for ODD inputs that are SAS tables or views. For details, see “Valid Inputs and Outputs for Data Stores” on page 55.)

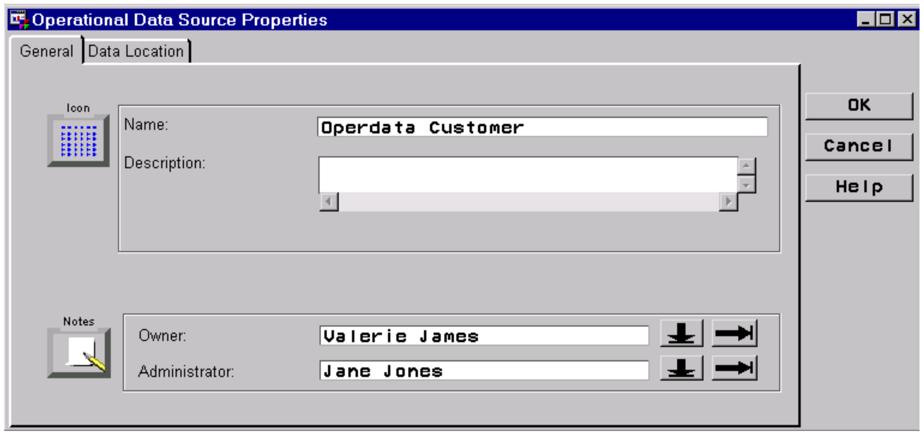
A default Data File is added to the Process Flow, as follows:



Next, update the default properties for the Data File.

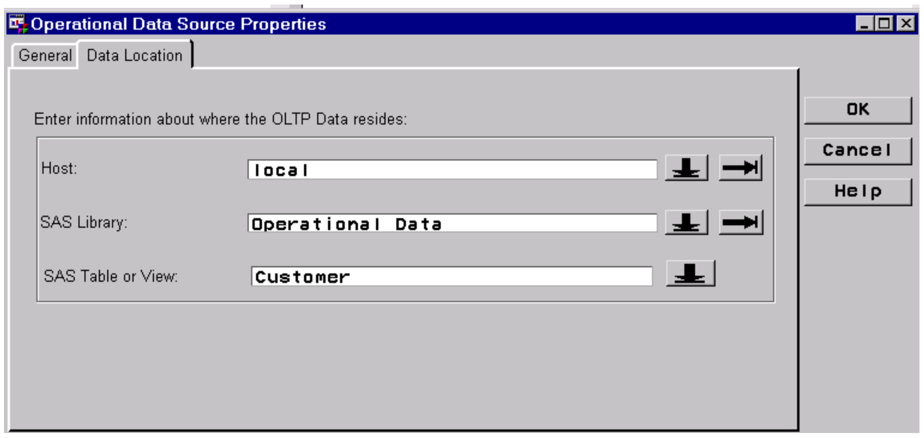
5 In the Process View of the Process Editor, position the cursor on the Data File, click the right mouse button and select **`Properties`**. The Operational Data Definition Process Attributes window displays for you to enter the appropriate information.

General Tab specifies the Data File's name, such as **Customer**, as well as a description, an owner, and an administrator.



Name: Operdata Customer
Description:
Owner: Valerie James
Administrator: Jane Jones

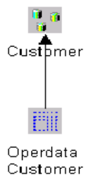
Data Location Tab specifies the definition for the host where the target library resides (such as **local**); specifies the target SAS library (such as **Operational Data**, the display name for the OPERDATA library); specifies the SAS table or view within that library (such as **Customer**). The next display shows an example of this metadata:



Enter information about where the OLTP Data resides:
Host: local
SAS Library: Operational Data
SAS Table or View: Customer

6 When you are finished entering metadata for the Data File, click OK to save it.

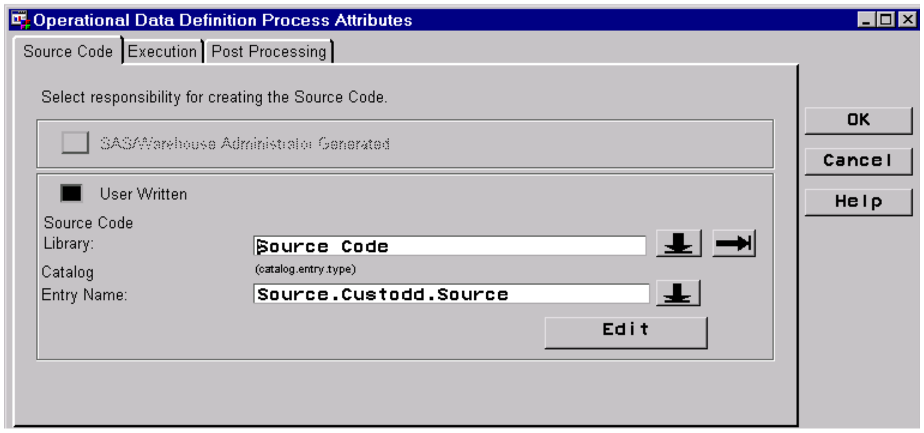
You are returned to the Process Editor. The Data File has its new name, if you specified one.



The Process Flow now specifies what information will flow from the source (Operdata Customer) to the target (Customer). The final task is to specify a Load Step that actually moves data from the source to the target.

7 In the Process View of the Process Editor, position the cursor on the Customer ODD, click the right mouse button and select **`Edit Load Step`**. The Operational Data Definition Process Attributes window displays for you to enter the appropriate information.

Source Tab specifies who supplies the source code for the Load process: you or SAS/Warehouse Administrator.



Select responsibility for creating the Source Code.

SAS/Warehouse Administrator Generated

User Written

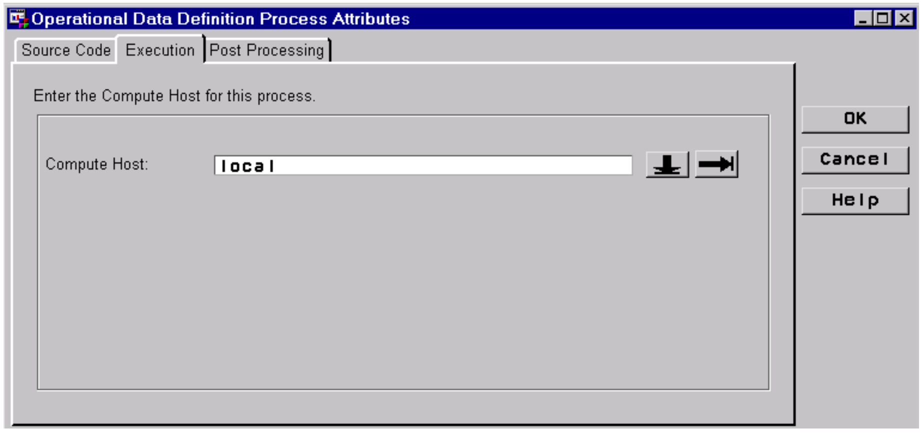
Source Code Library: Source Code

Catalog Entry Name: (catalog.entry.type) Source.Custodd.Source

The **`User Written`** option is selected automatically for ODDs. The **`Source Code Library`** field specifies the name of a SAS library that contains the routine to be executed: **`Source Code`** (the display name for the SOURCE library). The **`Catalog Entry Name`** field specifies a .SOURCE entry which contains the routine associated with the current Load Step: **`Source.custodd.source`**.

Next, specify the computer where the Load Step will run.

Execution Tab specifies the host on which you want to execute the Load process.



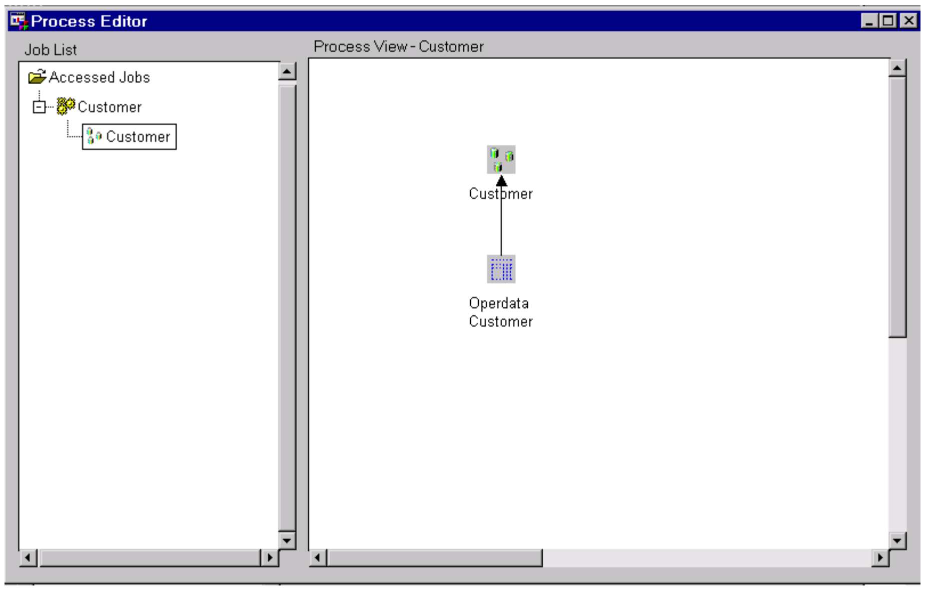
Enter the Compute Host for this process.

Compute Host: local

The **`Compute Host`** field specifies a definition for the host where the load step routine should be executed. The **`local`** host definition is specified in our example.

Post Processing Tab	specifies code to be executed after the Load process is finished. Leave this tab blank, for our example.

8 After you specify Execution tab metadata, you are finished creating this Load Step. Click OK to save its metadata record and return to the Process Editor.



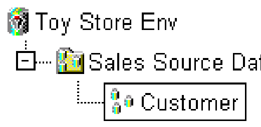
You are now ready to test the Job for this ODD.

Test the ODD

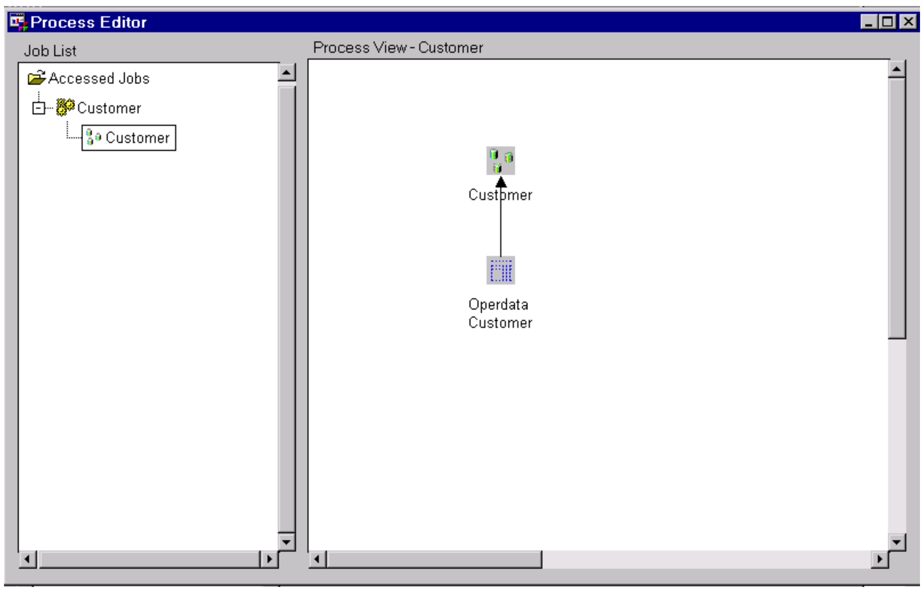
Execute the ODD Job

1 If you have not done so already, open the ODD in the Process Editor.

For example, display the ODD (such as Customer) in the SAS/Warehouse Administrator Explorer, as follows:

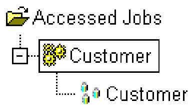


Position the cursor on the ODD, click the right mouse button and select **Process**. The ODD will be opened in the Process Editor, as follows:

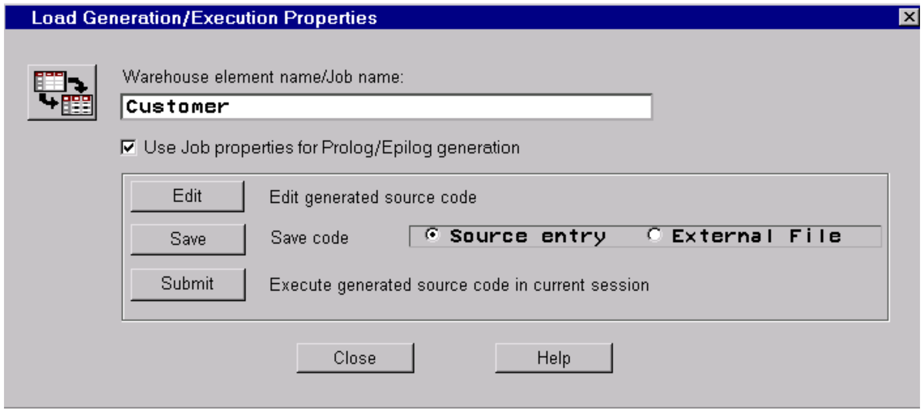


In the left panel of the Process Editor, the ODD's Job and output tables will be listed. In this display, the output table for the Job has a rectangle around it. Note that the output table has a parent. This parent is the Job for the Customer ODD.

2 (Optional) Position the cursor on the Job, and click the left mouse button to select it, as follows:



3 Position the cursor on the Job, click the right mouse button and select **Run**. The Load Generation/Execution Properties window displays.



Warehouse element name/Job name: Customer

☑ Use Job properties for Prolog/Epilog generation

Edit generated source code

Save code [✓ Source entry] External File

Execute generated source code in current session

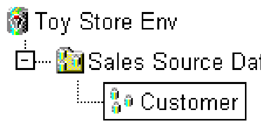
4 Click [Submit] to execute the Job.

Check the SAS log to verify that the Job was successful.

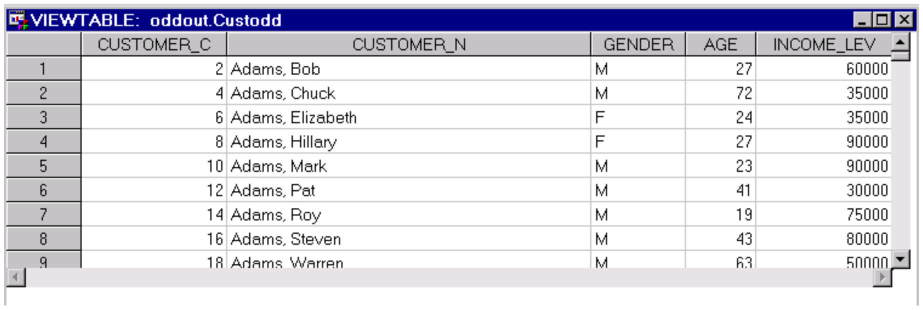
View the ODD Data

Use this method to view the data that is accessed through an ODD. In order for this method to work, the Job for an ODD must have been successfully executed at least once.

Display the ODD (such as Customer) in the SAS/Warehouse Administrator Explorer, as follows:



Position the cursor on the ODD, click the right mouse button and select **`Data Utilities`**, and then **`Open`**. The ODD will be opened in a VIEWTABLE window, as follows:



VIEWTABLE: oddout.Custodd

	CUSTOMER_C	CUSTOMER_N	GENDER	AGE	INCOME_LEV
1	2	Adams, Bob	M	27	60000
2	4	Adams, Chuck	M	72	35000
3	6	Adams, Elizabeth	F	24	35000
4	8	Adams, Hillary	F	27	90000
5	10	Adams, Mark	M	23	90000
6	12	Adams, Pat	M	41	30000
7	14	Adams, Roy	M	19	75000
8	16	Adams, Steven	M	43	80000
9	18	Adams, Warren	M	63	50000

After you have tested your ODD, it is ready to be used as an input to another data store. See also the maintenance note in “Keeping ODD Column Metadata Current” on page 130.

Keeping ODD Column Metadata Current

After you create an ODD, if any changes are made to the columns in the data source that is specified on the **`Data Location`** tab, you must update the ODD’s column metadata to match the data source.

Mapping processes, such as the one shown in Figure 7.1 on page 108, use the Columns tab metadata to map columns from the source to the target. The Columns tab metadata must be accurate, or the Mapping will not match the columns that are actually in the data source.

Accessing Data in ERP Systems

SAS provides these interfaces to Enterprise Resource Planning (ERP) systems:

- SAS/ACCESS Interface to R/3 from SAP AG
- SAS/ACCESS Interface to Baan
- SAS/ACCESS Interface to PeopleSoft

Contact your SAS technical support coordinator for details about accessing ERP data.

What's Next

After you have defined one or more ODDs, you are ready to add a Data Warehouse and a Subject. Then you can specify ODDs as inputs to data stores that you will add under a Subject area.

CHAPTER 8
Maintaining Data Warehouses and Subjects

Overview

After you have created a Warehouse Environment, you can define one or more Data Warehouses within it. In SAS/Warehouse Administrator, a Data Warehouse is a metadata record that specifies the SAS library _DWMD. The _DWMD library is the metadata repository for most of the groups and data stores in a data warehouse or a data mart at your site.

To create a Data Warehouse:

1 Open the appropriate Warehouse Environment in the Explorer.
2 Define the properties of the Data Warehouse.

After you have created a Data Warehouse, you can define one or more Subjects within it. A Subject is a grouping element for data related to one topic within a Data Warehouse. For example, a Data Warehouse might have a Subject called Products (information related to products) or Sales (information related to sales). Each Subject can be composed of a number of different data collections: detail data, summary data, charts, reports, and graphs.

To create a Subject:

1 Open the appropriate Warehouse Environment in the Explorer.
2 Locate the Data Warehouse in which the Subject will be added.
3 Add the Subject to the Data Warehouse.
4 Update the default properties of the Subject.

Note: The basic steps for maintaining Data Warehouses and Subjects are described in the online Help. To display the relevant online Help, in the SAS System Help contents, select

Help on SAS Software Products ▶ Using SAS/Warehouse Administrator Software

▶ Defining Data Warehouses and Subjects

In addition, you can display Help for most SAS/Warehouse Administrator windows by selecting the Help button on the window. △

Preparing to Create Local or Remote Data Warehouses

Before you create a Data Warehouse, you must do some preparation, which is summarized as follows:

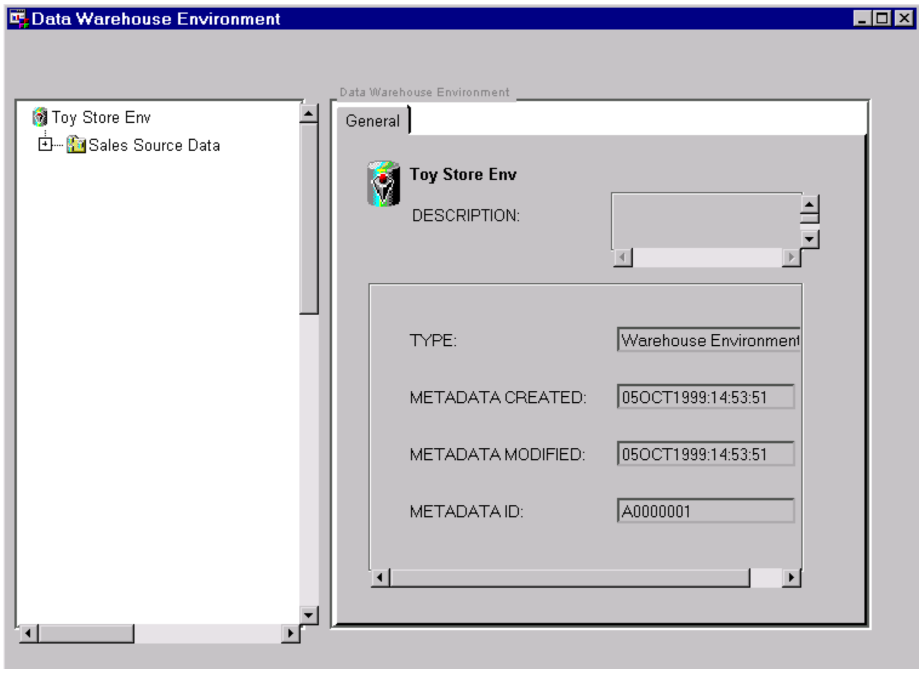
Metadata host configuration	□ Verify that you have the appropriate hardware and software to implement your Data Warehouse, as described in "Metadata Host Configuration" on page 23.
Hierarchy of objects	□ The Warehouse Environment that will contain the Data Warehouse must exist.
Global metadata	□ Create Contact records for the owners and administrators of the Data Warehouses and Subjects that you will create. For details about Contact records, see Chapter 6, "Maintaining Global Metadata," on page 75.
Physical path	□ The physical path for the Data Warehouse metadata library (libref _DWMD) must exist.

Example: Creating a Local Data Warehouse

This example summarizes how to create a local Data Warehouse—a Warehouse whose metadata repository is stored on the SAS/Warehouse Administrator host. The following explanations describe the metadata and methods used to achieve the desired results. It is assumed that the parent Environment exists.

Define Data Warehouse Properties

Open the appropriate Warehouse Environment in the Explorer, as described in "Opening a Warehouse Environment in the Explorer" on page 63. The following display illustrates what a new Warehouse Environment would look like after an ODD Group has been defined (as described in the "Registering Data Sources" chapter), but before any Data Warehouses have been defined:



Toy Store Env

DESCRIPTION:

TYPE: Warehouse Environment

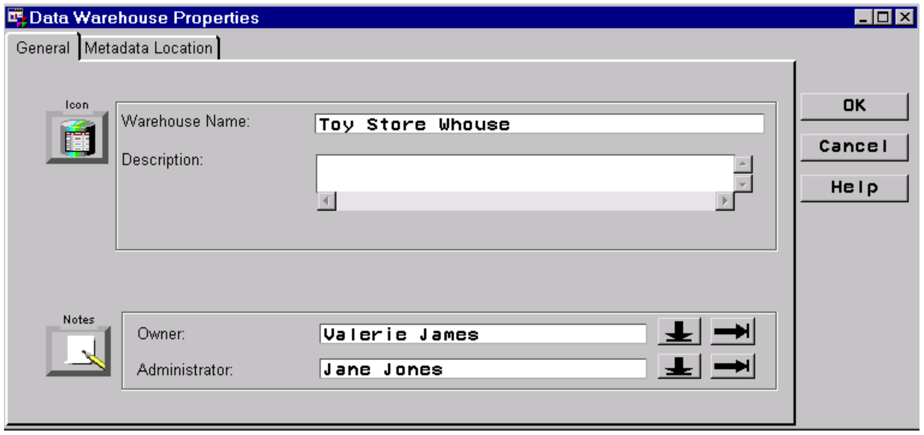
METADATA CREATED: 05OCT1999:14:53:51

METADATA MODIFIED: 05OCT1999:14:53:51

METADATA ID: A0000001

In the Explorer, position the cursor on the Environment, for example, **`Toy Store Env`**, click the right mouse button, select **`Add Item`**, and then **`Data Warehouse`**. A properties window for the Warehouse displays for you to enter the appropriate information.

General Tab specifies the Warehouse's name, such as **`Toy Store Whouse`**, as well as a description, an owner, and an administrator.



Warehouse Name: Toy Store Whouse

Description:

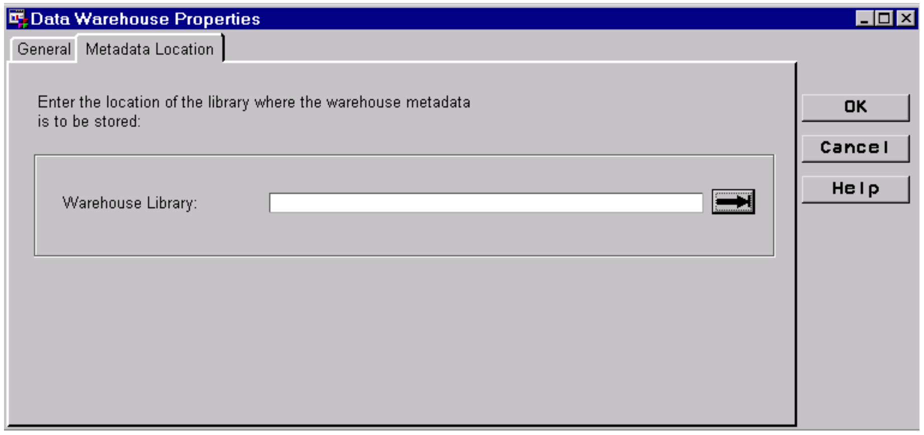
Owner: Valerie James

Administrator: Jane Jones

To specify an owner or an administrator, click the down arrow to select a name from a list. If you need to add a new Contact record, click the right arrow to display a properties window. For details about that window, click its [Help] button.

The next task is to specify the location of the metadata library for this Warehouse.

Metadata Location Tab
: specifies the metadata library (libref _DWMD) for the Warehouse. Before the metadata library has been defined, the **`Warehouse Library`** field is blank, as follows:

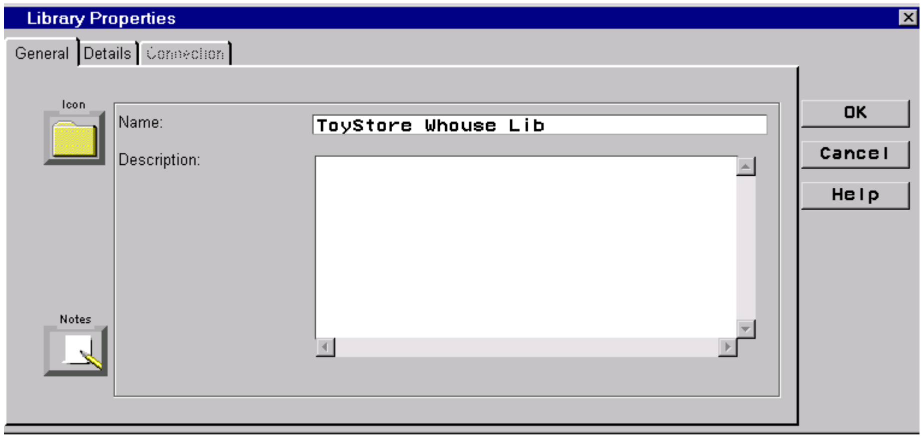


Enter the location of the library where the warehouse metadata is to be stored:

Warehouse Library:

On the Metadata Location tab, click the right arrow to define a metadata library for this Warehouse. A properties window for the Warehouse library displays for you to enter the appropriate information.

Name Tab
: specifies a display name for the local metadata library you are creating in this example, such as **`ToyStore Whouse Lib`**.

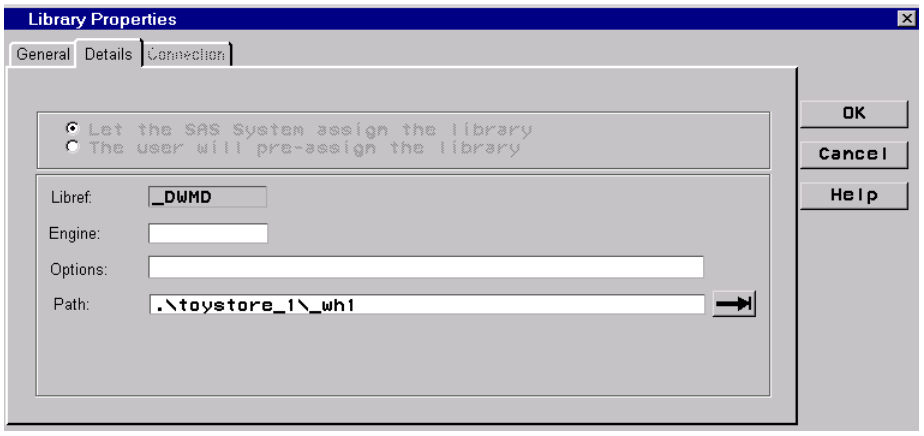


Name: ToyStore Whouse Lib

Description:

For a local metadata library, the next task is to specify the physical path for the library.

Details Tab
: specifies the physical path for the metadata library that you are creating, such as **`./toytstore_1/_wh1`**. For a local Data Warehouse, leave the **`Engine`** and **`Options`** fields blank, as follows:



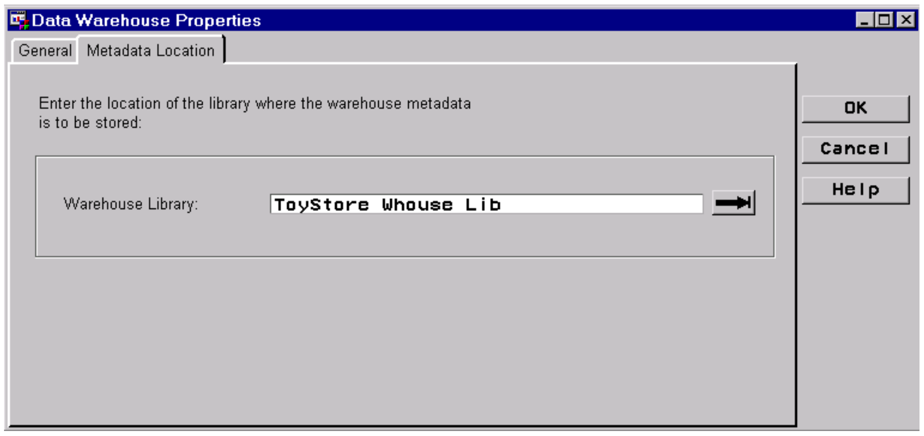
Let the SAS System assign the library [✓]
The user will pre-assign the library

Libref: _DWMD
Engine:
Options:
Path: .\toystore_1_wh1

When you have specified a path in the Path field, click OK to return to the Metadata Location tab for the Data Warehouse.

Metadata Location Tab

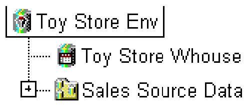
After defining the metadata library, the Metadata Location tab specifies the display name (**ToyStore Whouse Lib**) for the libref _DWMD:



Enter the location of the library where the warehouse metadata is to be stored:

Warehouse Library: ToyStore Whouse Lib

After specifying a name and metadata library for the Data Warehouse, click OK. The new Warehouse is added under the Environment in the Explorer, as follows:



You are finished entering properties for this local Data Warehouse. You are ready to add a Subject to the Warehouse.

Example: Creating a Remote Data Warehouse

This example summarizes how to create a remote Data Warehouse—a Warehouse whose metadata repository is not stored on the SAS/Warehouse Administrator host.

A remote Data Warehouse might be appropriate if you require concurrent read/write access to the Warehouse by multiple SAS/Warehouse Administrator hosts. In that case, you must create a remote Warehouse Environment and put its metadata repository—and the metadata repositories of its Data Warehouses—under the control of a SAS/SHARE server remote to the SAS/Warehouse Administrator hosts. For an example of such a configuration, see "Remote Metadata: PC Client to Windows NT Server" on page 25.

The following sections describe the metadata and methods used to achieve the desired results. In addition to the preparation described in "Preparing to Create Local or Remote Data Warehouses" on page 134, some additional preparation for remote Environments is described next.

SAS/SHARE Server Preparation

On the SAS/SHARE server host

- □ create a physical path for the Data Warehouse
- □ assign a SAS libref to the directory that corresponds to the _DWMD library for the Warehouse; do this in such a way that the library is under the control of the SAS/SHARE server.

For example, suppose that you created the following directory structure on the SAS/SHARE server host:

```
.\Project-2\_env
.\Project-2\_wh1
```

where **`.\Project-2\_wh1`** is the directory that corresponds to the _DWMD metadata repository for the Data Warehouse that you will create. You would then assign a libref to that directory in such a way that the library is under the control of the SAS/SHARE server.

Note: Because this is a SAS/SHARE libref and not a SAS/Warehouse Administrator libref, you are not restricted to _DWMD as the libref. For example, to the SAS/SHARE server, the **`.\Project-2\_wh1`** directory could have a SAS/SHARE libref of WH1. (The SLIBREF option would then have to be used in the Data Warehouse's metadata, as described in "Define Data Warehouse Properties" on page 139. △

Note: If you use a relative pathname such as **`.\Project-2\_wh1`** for the Warehouse library, be sure that the SAS/SHARE server can resolve the pathname. △

SAS/Warehouse Administrator Client Preparation

Verify that your local SAS session can access the remote library that will ultimately become the metadata repository for the Data Warehouse that you are creating. One way to do that is to submit a LIBNAME statement for that library. Here is an example of such a statement:

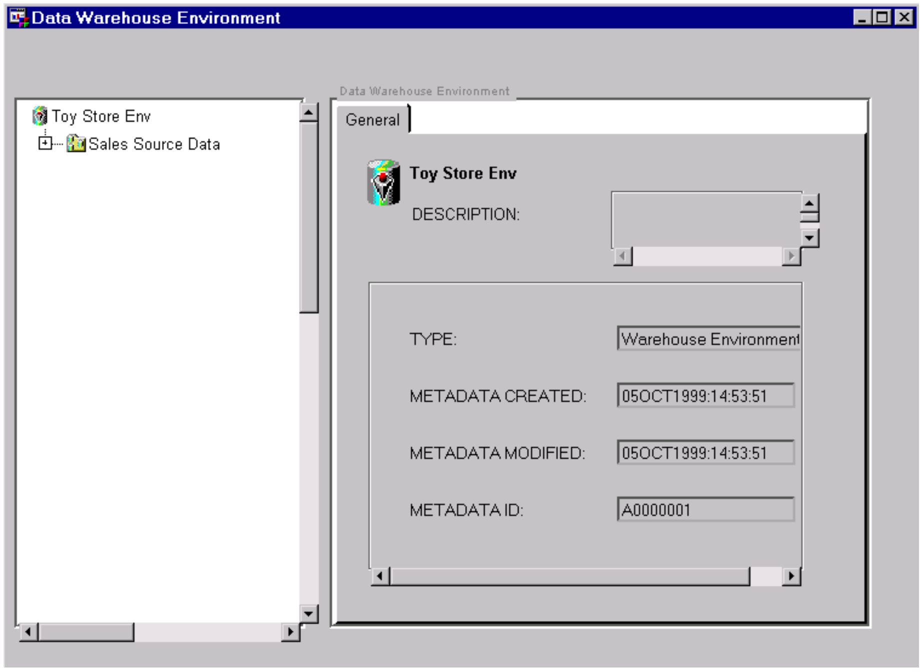
```
libname _DWMD server=host2.dwshare slibref=WH1;
```

where **`_DWMD`** is the local libref for the Data Warehouse metadata repository, **`host2`** is the name of the remote host, **`dwshare`** is the name of the SAS/SHARE server, and **`slibref=WH1`** is the remote server's libref for the Data Warehouse's metadata repository.

If you can execute such a statement successfully, then SAS/Warehouse Administrator can generate code successfully for the remote Data Warehouse.

Define Data Warehouse Properties

Open the appropriate Warehouse Environment in the Explorer, as described in "Opening a Warehouse Environment in the Explorer" on page 63. The following display illustrates what a new Warehouse Environment would look like after an ODD Group has been defined (as described in the "Registering Data Sources" chapter), but before any Data Warehouses have been defined:



Toy Store Env

DESCRIPTION:

TYPE: Warehouse Environment

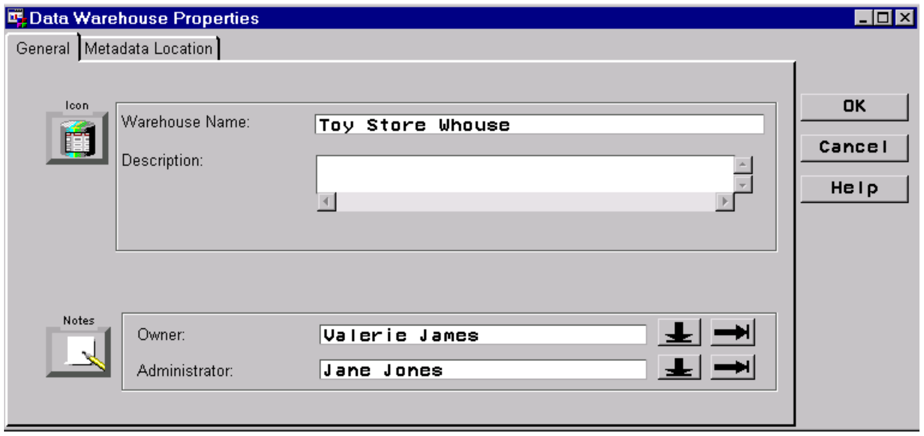
METADATA CREATED: 05OCT1999:14:53:51

METADATA MODIFIED: 05OCT1999:14:53:51

METADATA ID: A0000001

In the Explorer, position the cursor on the Environment, for example, **`Toy Store Env`**, click the right mouse button, select **`Add Item`**, and then **`Data Warehouse`**. A properties window for the Warehouse displays for you to enter the appropriate information.

General Tab	specifies the Warehouse name, such as **`Toy Store Whouse`**, as well as a description, an owner, and an administrator.

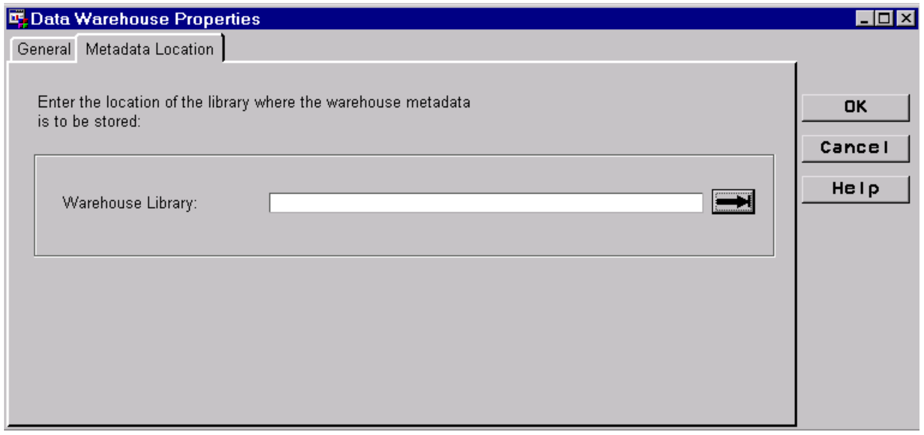


Warehouse Name: Toy Store Whouse
Description:
Owner: Valerie James
Administrator: Jane Jones

To specify an owner or an administrator, click the down arrow to select a name from a list. If you need to add a new Contact record, click the right arrow to display a properties window. For details about that window, click its Help button.

The next task is to specify the location of the metadata library for this Warehouse.

Metadata Location Tab
specifies the metadata library (libref _DWMD) for the Warehouse. Before the metadata library has been defined, the **`Warehouse Library`** field is blank, as follows:

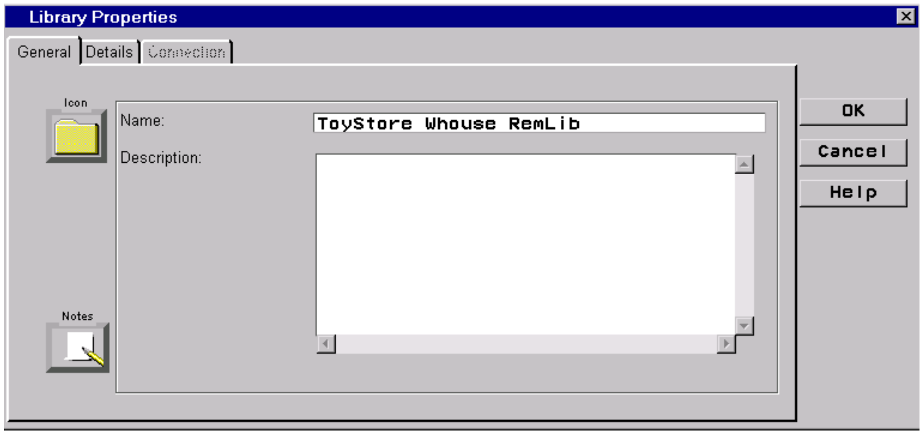


Enter the location of the library where the warehouse metadata is to be stored:

Warehouse Library:

On the Metadata Location tab, click the right arrow to define a metadata library for this Warehouse. A properties window for the Warehouse library displays for you to enter the appropriate information.

Name Tab
specifies a display name for the remote metadata library you are creating in this example, such as **`ToyStore Whouse RemLib`**.



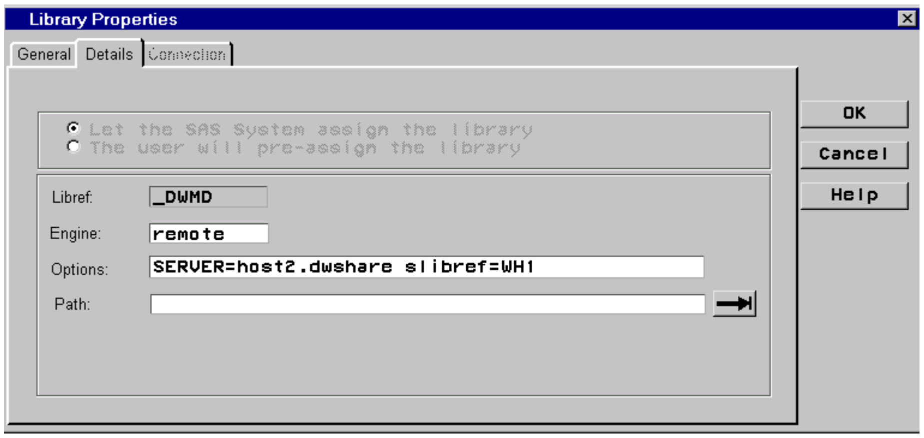
Name: ToyStore Whouse RemLib

Description:

For a remote metadata library, the next task is to specify the SAS engine and the options required to connect to the SAS/SHARE server.

Details Tab

For a remote metadata library, this tab specifies the SAS engine and the LIBNAME statement options required to connect to the SAS/SHARE server, as follows:



[✓ Let the SAS System assign the library] The user will pre-assign the library

Libref: _DWMD

Engine: remote

Options: SERVER=host2.dwshare slibref=WH1

Path:

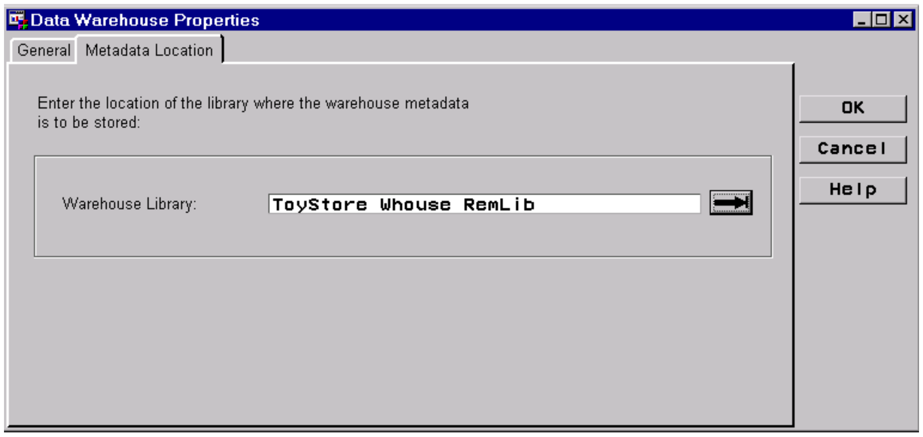
For the current example, the **Engine** field specifies that the **remote** SAS engine will be used to access the _DWMD library for the Data Warehouse.

The **Options** field specifies the SERVER= option required to access the _DWMD library for the Data Warehouse, where **host2** is the name of the remote host, **dwshare** is the name of the SAS/SHARE server, and **slibref=WH1** is the remote server's libref for the Data Warehouse's metadata repository.

Do not enter a physical path in the **Path** field because the _DWMD library will be accessed through the SAS/SHARE server, which uses a libref to access this library.

After specifying the appropriate values, click OK to return to the Metadata Location tab for the Data Warehouse.

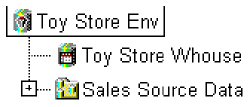
Metadata Location Tab	After defining the metadata library, the Metadata Location tab specifies the display name (**`ToyStore Whouse RemLib`**) for the libref _DWMD:



Enter the location of the library where the warehouse metadata is to be stored:

Warehouse Library: ToyStore Whouse RemLib

After specifying a name and metadata library for the Data Warehouse, click OK. The new Warehouse is added under the Environment in the Explorer, as follows:



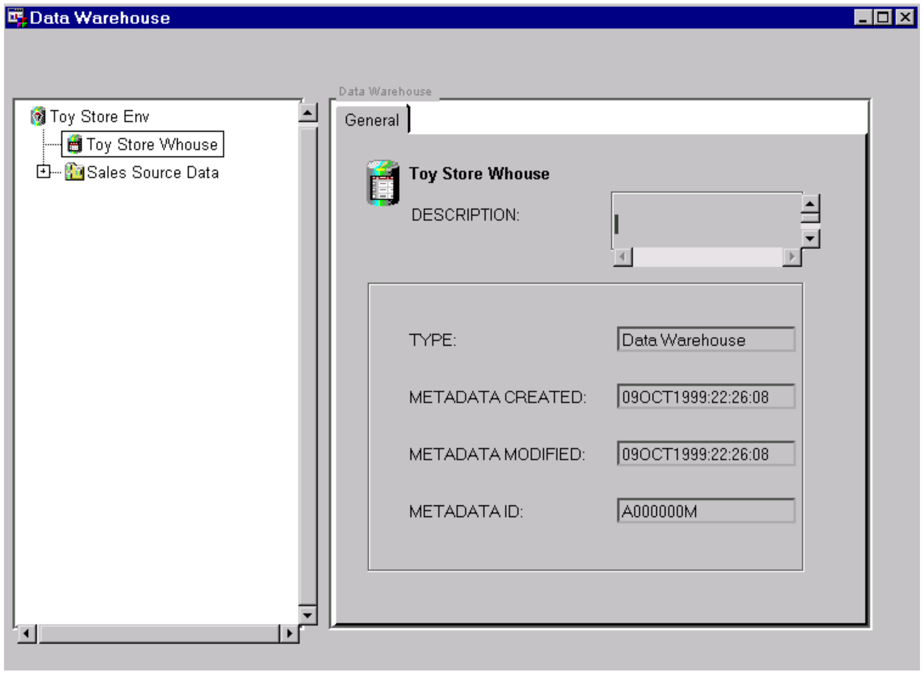
You are finished entering properties for this remote Data Warehouse. You are ready to add a Subject to the Warehouse.

Example: Creating a Subject

Your data warehouse project plan should specify a number of subject areas—such as finance, sales, marketing, and accounting—for which you will develop data collections. For each one of these areas, you will create a Subject in a Data Warehouse in the SAS/Warehouse Administrator Explorer.

Define Subject Properties

With the appropriate Warehouse Environment open in the Explorer, locate the Data Warehouse in which you will add the Subject. The next display illustrates what a new Data Warehouse (**`Toy Store Whouse`**) would look like before any Subjects have been defined:

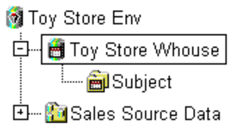


Toy Store Whouse

DESCRIPTION:

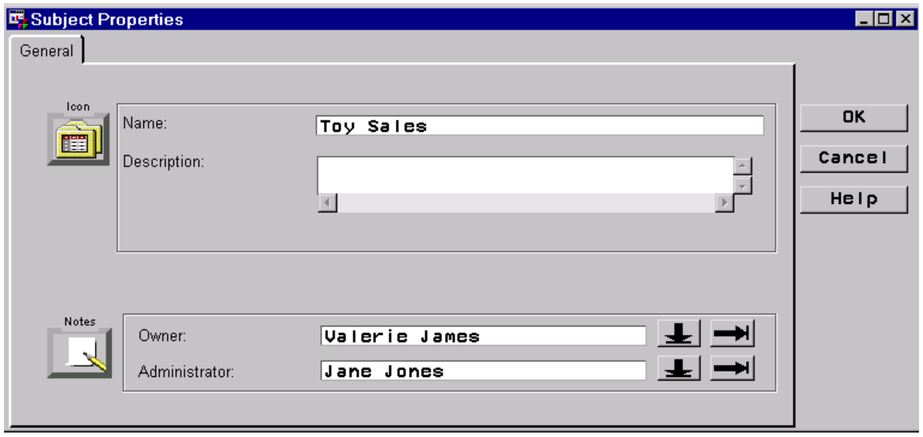
TYPE:	Data Warehouse
METADATA CREATED:	09OCT1999:22:26:08
METADATA MODIFIED:	09OCT1999:22:26:08
METADATA ID:	A000000M

In the Explorer, position the cursor on the Data Warehouse, for example, **`Toy Store Whouse`**, click the right mouse button, select **`Add Item`**, and then **`Subject`**. In the Explorer window, a new Subject is added under the Data Warehouse, as follows:



To update the default metadata for the Subject, position the cursor on its icon, click the right mouse button, and select **`Properties`**. A properties window for the Subject displays for you to enter the appropriate information.

General Tab specifies the Subject name, such as **`Toy Sales`**, as well as a description, an owner, and an administrator.



Name:	Toy Sales
Description:	
Owner:	Valerie James
Administrator:	Jane Jones

To specify an owner or an administrator, click the down arrow to select a name from a list. If you need to add a new Contact record, click the right arrow to display a properties window. For details about that window, click its $\boxed{\text{Help}}$ button.

After specifying information on this tab, you are finished creating the Subject. Click $\boxed{\text{OK}}$ to save its metadata record and return to the Explorer.

What's Next

After adding a Data Warehouse and a Subject, you are ready to define detail data stores in the Subject.

CHAPTER

9

Maintaining Data Tables

Overview

This chapter assumes that you have evaluated the different detail data stores available in SAS/Warehouse Administrator, as described in "Detail Data Stores" on page 46, and have chosen to create Data Tables. SAS/Warehouse Administrator provides the Data Table as a multipurpose data store, providing flexibility and usability because of few restrictions regarding hierarchical placement and types of input sources. In SAS/Warehouse Administrator, the following objects are provided for multiple purposes:

Data Group	is a multipurpose, simple grouping element that allows you to organize Data Tables, other Data Groups, and Information Marts. You can add a Data Group to a Warehouse, Subject, or another Data Group.
Data Table	is a metadata record that specifies a SAS table or view or a DBMS table or view that can serve multiple purposes. You can use Data Tables as intermediate data stores (such as look-up tables), 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. You can add a Data Table to a Data Group only. *Note:* You can also store detail data in Detail Tables. However, Data Tables are more multipurpose. Data Table metadata is the same as Detail Table metadata, except that it includes the SAS code (such as a VIEWTABLE statement) used to read the Data Table when you open it from a data utility. △

In general, to maintain Data Tables:

1 Define the properties for the Data Group.

2. Define the properties for the Data Table(s).
3. Define a Process Editor Job that includes the data preparation processes, which prepares the data to be loaded into the Data Tables, and the Load Step, which defines the steps to load the Data Tables.
4. Execute the Job.
5. Verify that the Data Tables are loaded; that is, check logs, use the data utilities, and so on.

Note: The basic steps for creating a Data Group and a Data Table are described in the online Help. To display the relevant online Help, in the SAS System Help contents, select

Help on SAS Software Products ▶ Using SAS/Warehouse Administrator Software ▶ Defining Detail Data Stores ▶ Maintaining Data Tables

In addition, you can display Help for most SAS/Warehouse Administrator windows by selecting the Help button on the window. △

Preparing to Create Data Tables

Before you create a Data Group and its associated Data Tables, you must first do some preparation, as follows:

Hierarchy of objects	□ In SAS/Warehouse Administrator, make sure that you have created the appropriate Data Warehouse Environment, Data Warehouse, and Subject.
Input source(s)	□ Determine which input sources(s) to use for each Data Table. For example, if the input source for a Data Table is an ODD, the ODD must be fully operational.
	□ Analyze the input data and determine which columns you want to include in the Data Table.
	□ Decide what columns are needed for the Data Table.
Processes	□ Determine whether the source code, which defines how data moves from input source(s) to output targets, is generated by SAS/Warehouse Administrator or if is it user written and stored in a SAS catalog entry.
	□ Determine the type of Mapping processes, for example, you might need to transform data or generate data.
	□ Determine whether you need to define other processes such as a User Exit process, Data Transfer process, or Record Selector process.
Output target	□ Determine how you want to store each Data Table. For example, determine what format you want to store the data, such as SAS format or a DBMS.
	□ Determine on what platform you want to store the Data Table. That is, you can store the table locally or on a remote host.

Example: Creating a Data Group

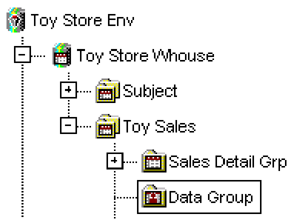
Overview

The Data Group is a simple grouping element that you create to group a variety of data stores, which include Data Tables. This example creates a Data Group to group related Data Tables.

Note: The following explanations describe the metadata and methods used to achieve the desired results; it is assumed that the appropriate Data Warehouse Environment, Data Warehouse, and Subject exist. △

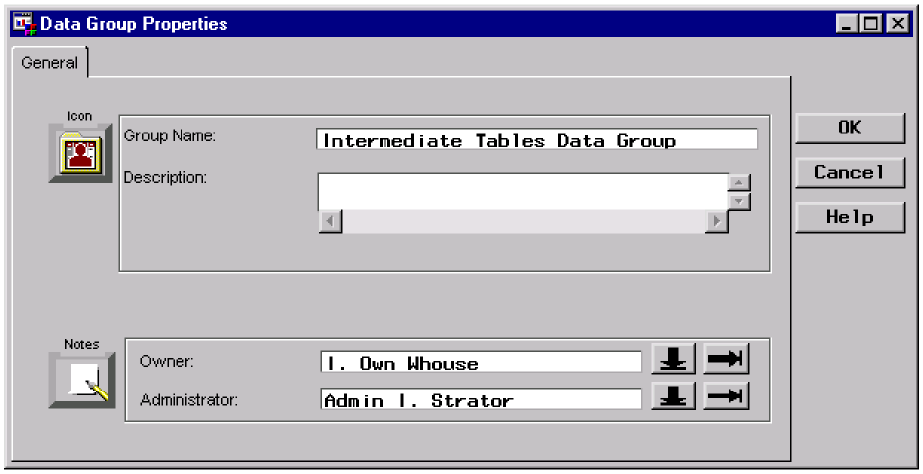
Define Data Group Properties

In the SAS/Warehouse Administrator Explorer, position the cursor on the Subject, for example, **`Toy Sales`**, click the right mouse button, select **`Add Item`**, and then **`Data Group`**. In the Explorer window, a new Data Group is added under the Subject as follows:



To update the default metadata for the Data Group, position the cursor on its icon, click the right mouse button, and select **`Properties`**. The Data Group Properties window displays for you to enter the appropriate information.

General Tab specifies the group name, for example, **`Intermediate Tables Data Group`**, a description, an owner, and an administrator.



Group Name: Intermediate Tables Data Group
Description:
Owner: I. Own Whouse
Administrator: Admin I. Strator

The next section describes how to create a Data Table to add to the Data Group.

Example: Creating a Data Table

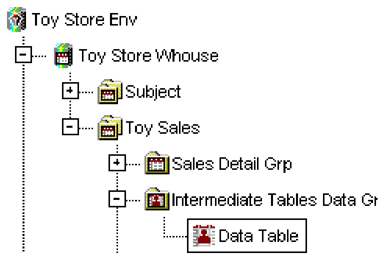
Overview

After you have a Data Group to group Data Tables, you can then add Data Tables to that group. This example creates a Data Table to be used as a look-up table, for example, to find the city and state associated with a specific zipcode.

Note: The following explanations describe the metadata and methods to achieve the desired results; it is assumed that the appropriate Data Warehouse Environment, Data Warehouse, Subject, Data Group, and ODD exist. △

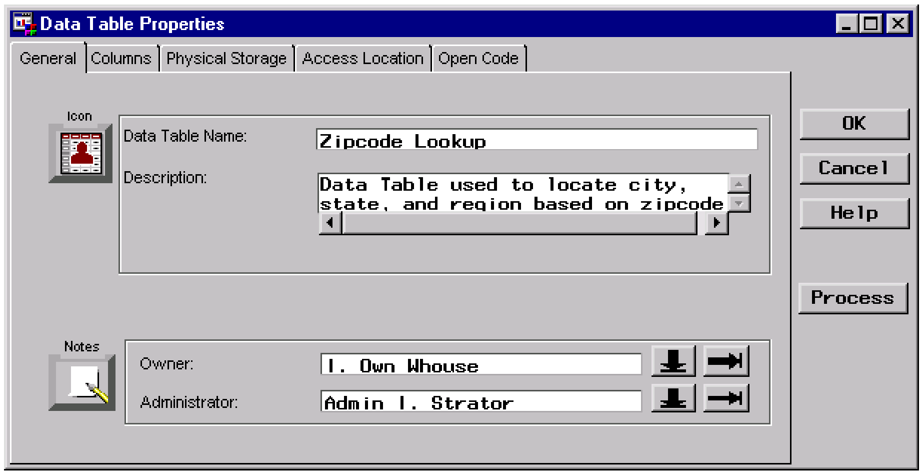
Define Data Table Properties

In the SAS/Warehouse Administrator Explorer, position the cursor on the Data Group, for example, **`Intermediate Tables Data Group`**, click the right mouse button, select **`Add Item`**, and then **`Data Table`**. In the Explorer window, a new Data Table is added under the Data Group as follows:


```
Toy Store Env
  Toy Store Whouse
    Subject
    Toy Sales
      Sales Detail Grp
      Intermediate Tables Data Gr
        Data Table
```

To update the default metadata for the table, put the cursor on its icon, press the right mouse button, and select **`Properties`**. The Data Table Properties window displays for you to enter the appropriate information, which is described next:

General Tab — specifies the table's name **`Zipcode Lookup`**, a description, an owner, and an administrator.



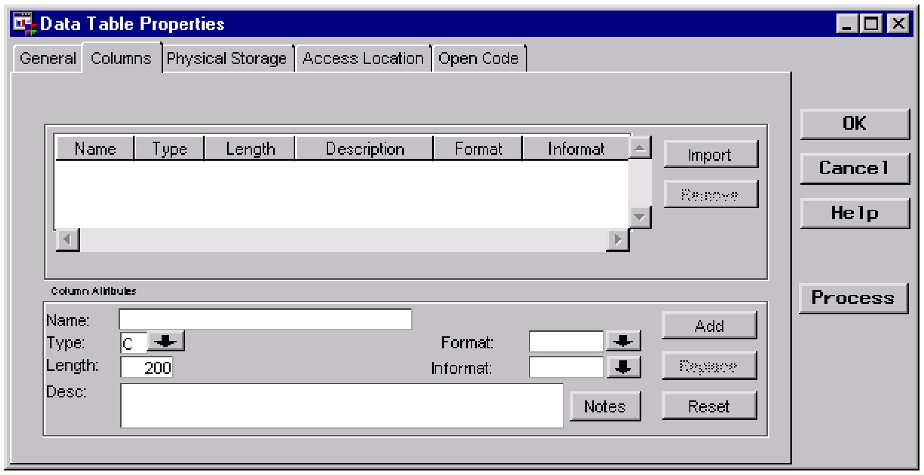
Data Table Name: Zipcode Lookup

Description: Data Table used to locate city, state, and region based on zipcode

Owner: I. Own Whouse

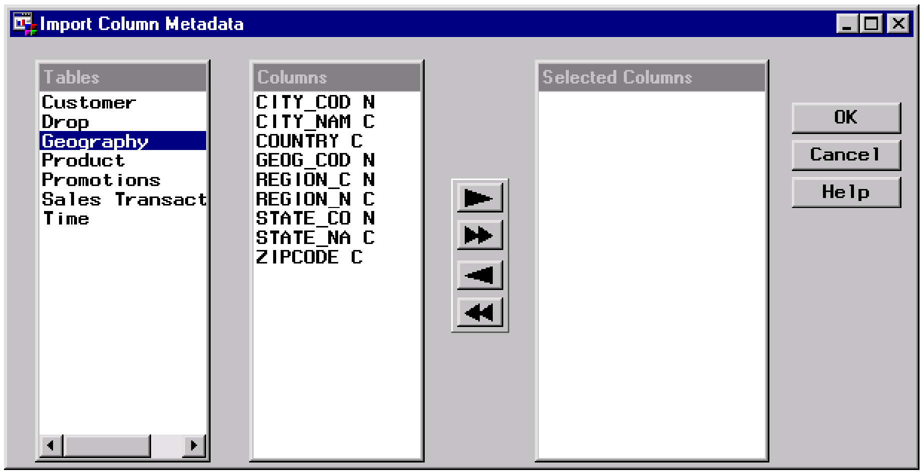
Administrator: Admin I. Strator

Columns Tab specifies the columns to be included in the Data Table, which do not exist yet as follows:

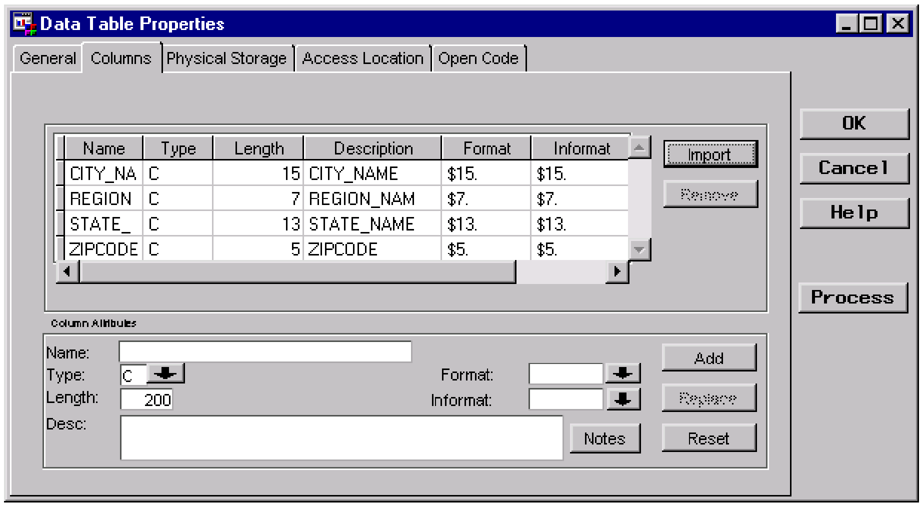


To import columns from an input source, click Import to display a list of input sources, and then select one, for example, **`Operational Data Sources`**. The Import Column Metadata window displays.

From the Import Column Metadata window, which lists the available sources, select an ODD, for example **`Geography`**, to display its available columns.



Select the appropriate columns under **`Columns`** (which for this example are **`city_nam`**, **`region_n`**, **`state_na`**, and **`zipcode`**), use the double arrows to move them to **`Selected Columns`**, and then click OK. You are returned to the Columns tab in the Data Table Properties window, which lists the imported columns:


Data Table Properties

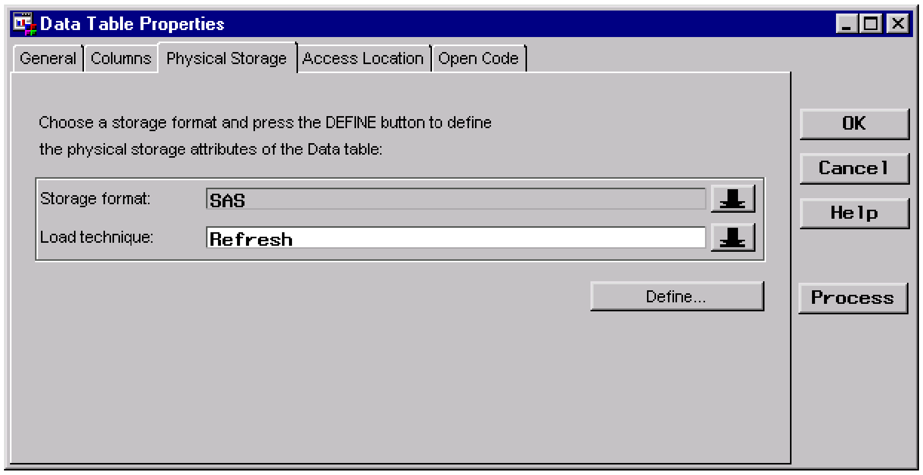
General | Columns | Physical Storage | Access Location | Open Code

Name	Type	Length	Description	Format	Informat
CITY_NA	C	15	CITY_NAME	$15.	$15.
REGION	C	7	REGION_NAM	$7.	$7.
STATE_	C	13	STATE_NAME	$13.	$13.
ZIPCODE	C	5	ZIPCODE	$5.	$5.

Column Attributes

Name:
Type: C
Length: 200
Desc:
Format:
Informat:

Physical Storage Tab specifies the physical storage attributes. This example specifies the storage format **SAS** and the load technique **Refresh**.


Data Table Properties

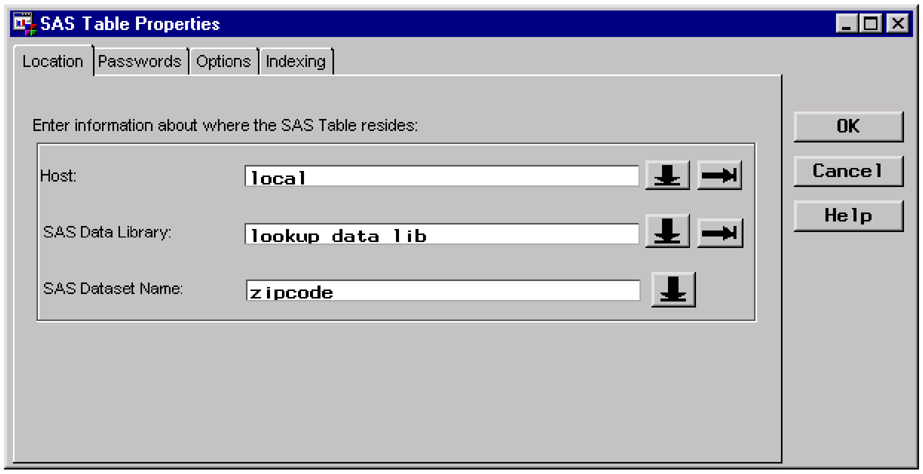
General | Columns | Physical Storage | Access Location | Open Code

Choose a storage format and press the DEFINE button to define the physical storage attributes of the Data table:

Storage format: SAS
Load technique: Refresh

Click Define to open the SAS Table Properties window and view its metadata.

Location Tab specifies where the SAS table is stored.


SAS Table Properties

Location | Passwords | Options | Indexing

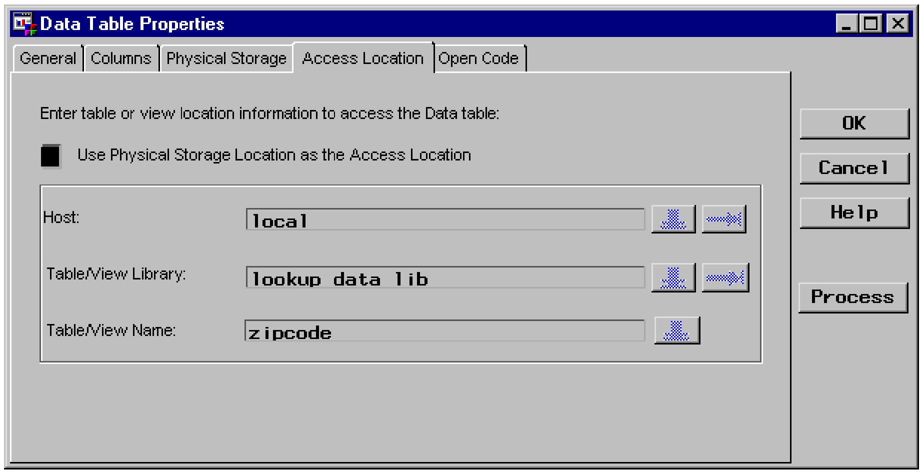
Enter information about where the SAS Table resides:

Host: local

SAS Data Library: lookup data lib

SAS Dataset Name: zipcode

Access Location Tab	specifies a location to provide interactive access to the Data Table. For this example, the check box **`Use Physical Storage Location as the Access Location`** is selected by default, which enables the software to use information from the Physical Storage tab to provide interactive access to the table.


Data Table Properties

General | Columns | Physical Storage | Access Location | Open Code

Enter table or view location information to access the Data table:

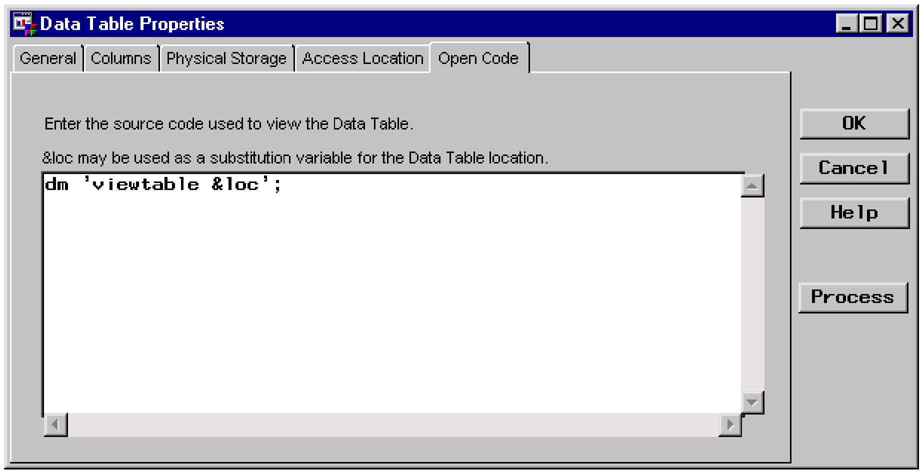
Use Physical Storage Location as the Access Location

Host: local

Table/View Library: lookup data lib

Table/View Name: zipcode

Open Code Tab	specifies the source code used to view the table.



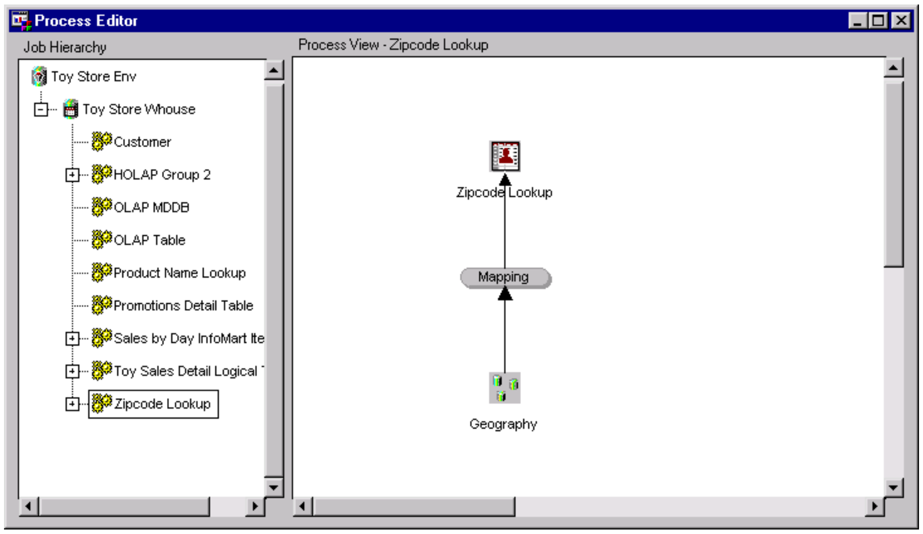
Enter the source code used to view the Data Table.

&loc may be used as a substitution variable for the Data Table location.

```
dm 'viewtable &loc';
```

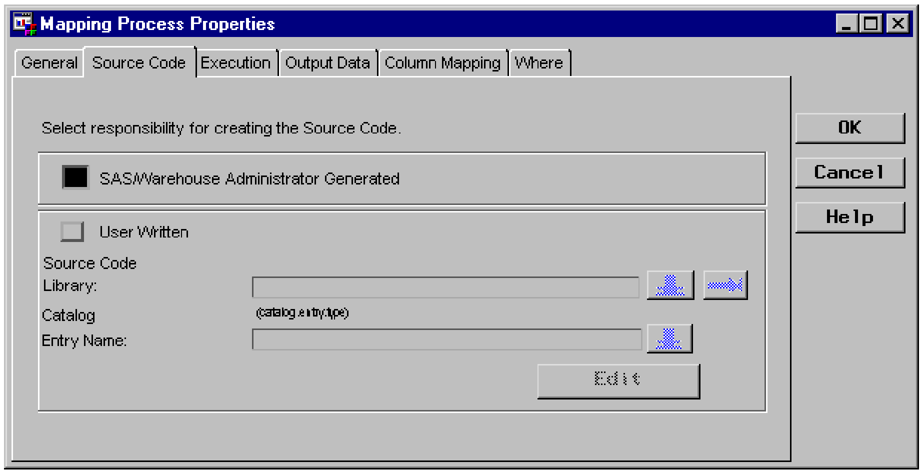
Define Process Editor Job

In the Process Editor Job, the Data Table **Zipcode Lookup** is specified as the output target and the ODD **Geography** is specified as the input source. The following Process Editor window shows the Process Flow for the Job:



The processes defined in the Job are summarized as follows:

Mapping Process	□ The source code to map columns is generated by SAS/Warehouse Administrator, rather than user written, as shown in the Source Code tab:


Mapping Process Properties

General | Source Code | Execution | Output Data | Column Mapping | Where

Select responsibility for creating the Source Code.

■ SAS/Warehouse Administrator Generated

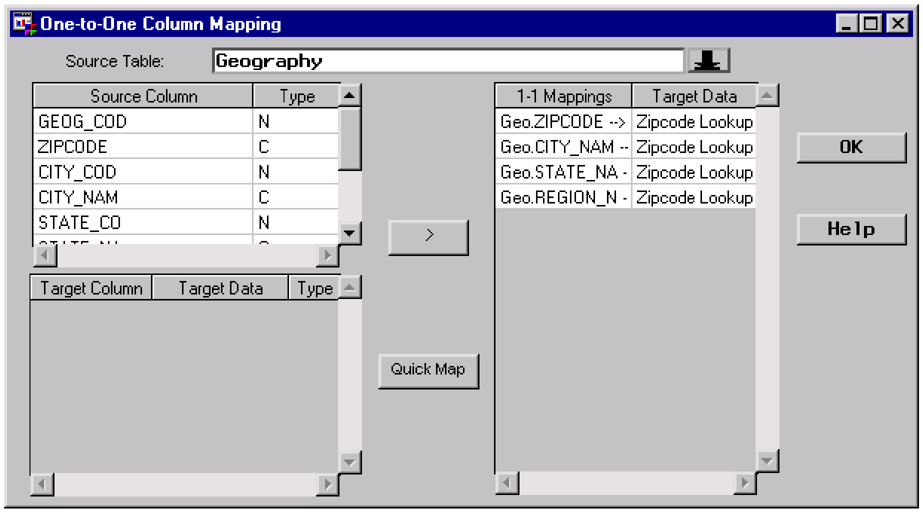
☐ User Written

Source Code
Library:
Catalog (catalog.entry.type)
Entry Name:

Edit

OK | Cancel | Help

□ Column mapping is defined as one-to-one mapping, as shown in the Column Mapping tab. To produce one-to-one mapping, first click $\boxed{\text{1 to 1 Mappings}}$ from the Column Mapping tab, which opens the One-to-One Column Mapping window. Then, click $\boxed{\text{Quick Map}}$.


One-to-One Column Mapping

Source Table: Geography

Source Column	Type
GEOG_COD	N
ZIPCODE	C
CITY_COD	N
CITY_NAM	C
STATE_CO	N

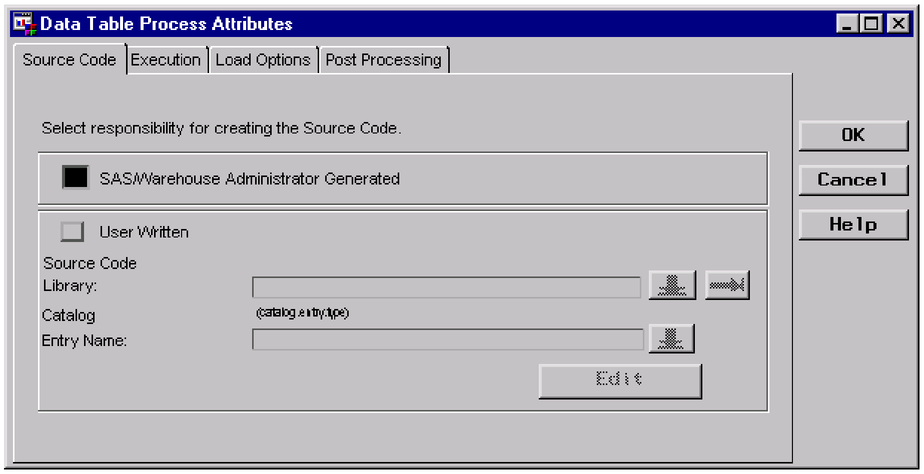
Target Column	Target Data	Type

1-1 Mappings	Target Data
Geo.ZIPCODE -->	Zipcode Lookup
Geo.CITY_NAM --	Zipcode Lookup
Geo.STATE_NA -	Zipcode Lookup
Geo.REGION_N -	Zipcode Lookup

> | Quick Map | OK | Help

Load Step Process

□ The source code is generated by SAS/Warehouse Administrator, rather than user written, as shown in the Data Table Load Process Attributes window:



Select responsibility for creating the Source Code.

■ SAS/Warehouse Administrator Generated

☐ User Written

Source Code Library:

Catalog Entry Name: (catalog.entry.type)

For more information about Process Editor Jobs, see Chapter 13, "Maintaining Jobs," on page 251. For more information about processes, see Chapter 14, "Maintaining Processes," on page 281.

What's Next

After you create Data Tables, you can use them as inputs to other SAS/Warehouse Administrator objects such as OLAP summary data stores and Detail Tables, or you can exploit them with tools designed to work with detail data, such as data mining applications.

CHAPTER

10

Maintaining Detail Logical Tables and Detail Tables

Overview

This chapter assumes that you have evaluated the different detail data stores available in SAS/Warehouse Administrator, as described in Chapter 4, "Planning Your Data Stores and Processes," on page 41, and have chosen to create these data stores:

Detail Logical Table	is a metadata record that specifies a SAS table or view that can serve multiple purposes. You can use a Detail Logical Table as a grouping element for Detail Tables or as an individual detail data store, for example, as a view to multiple, related Detail Tables. You can add a Detail Logical Table to a Subject only, and a Subject can have only one Detail Logical Table.
Detail Table	is a metadata record that specifies a SAS table or view or a DBMS table or view that serves as a detail data store. You can add a Detail Table to a Detail Logical Table only.

In general, to maintain a Detail Logical Table and associated Detail Tables:

1 Define the properties of the Detail Logical Table.

2 Define the properties of the Detail Table(s).

3 Define a Process Editor Job that includes the data preparation processes, which prepare the data to be loaded into the data store(s), and the Load Step, which defines the steps to load the data store(s).

4 Execute the Job.

5 Verify that the data store(s) are loaded; that is, check logs, use the data utilities, and so on.

Note: The basic steps for creating a Detail Logical Table and Detail Tables are described in the online Help. To display the relevant online Help, in the SAS System Help contents, select

Help on SAS Software Products ▶ Using SAS/Warehouse Administrator Software

▶ Defining Detail Data Stores ▶ Maintaining Detail Logical Tables or

▶ Maintaining Detail Tables

In addition, you can display Help for most SAS/Warehouse Administrator windows by selecting the Help button on the window. △

Preparing to Create Detail Tables

Before you create a Detail Logical Table and its associated Detail Tables, you must first do some preparation as follows:

Hierarchy of objects	□ In SAS/Warehouse Administrator, make sure that you have created the appropriate Data Warehouse Environment, Data Warehouse, and Subject.
Input source(s)	□ Determine which input source(s) to use for each Detail Table. For example, if the input source for a Detail Table is an ODD, the ODD must be fully operational.
	□ Analyze the input data and determine which columns you want to include in the Detail Table.
	□ Decide what columns are needed for the Detail Table.
Processes	□ Determine whether the source code, which defines how data moves from input sources to output targets, is generated by SAS/Warehouse Administrator or if is it user written and stored in a SAS catalog entry.
	□ Determine the type of Mapping processes, for example, you might need to transform data or generate data.
	□ Determine whether you need to define other processes such as a User Exit process, Data Transfer process, or Record Selector process.
Output target	□ Determine how you want to store each Detail Table. For example, determine what format you want to store the data, such as SAS format or a DBMS.
	□ Determine on what platform you want to store the Detail Table. That is, you could store the table locally or on a remote host.

Example: Creating a Detail Logical Table as a Grouping Element for Detail Tables

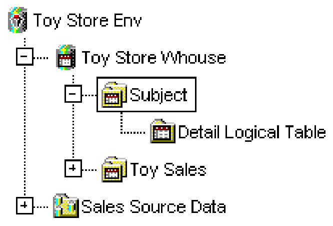
Overview

The simplest use of a Detail Logical Table is to create it as a grouping element for Detail Tables. Note that when a Detail Logical Table is used as a grouping element, it does not have an associated Process Editor Job. However, each Detail Table grouped by the Detail Logical Table would have a Process Editor Job.

Note: The following explanations describe the metadata and methods used to achieve the desired results; it is assumed that the appropriate Data Warehouse Environment, Data Warehouse, and Subject exist. △

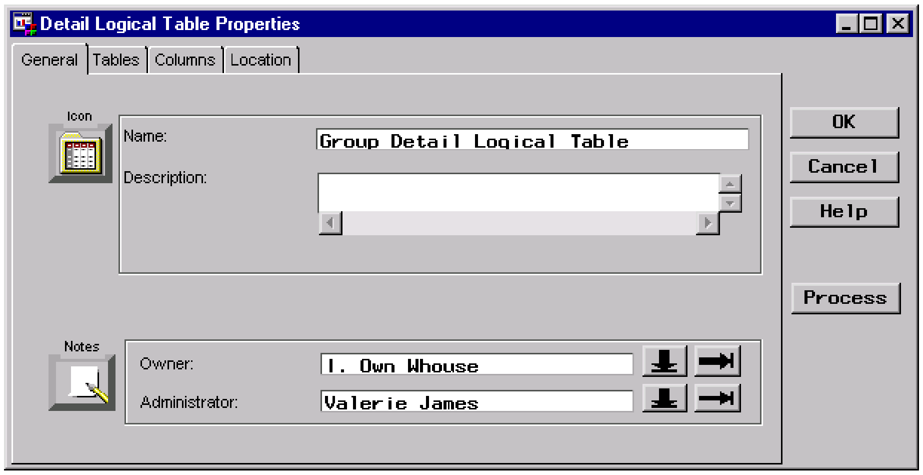
Define Detail Logical Table Properties

In the SAS/Warehouse Administrator Explorer, position the cursor on the Subject, for example **`Subject`**, click the right mouse button, select **`Add Item`**, and then **`Detail Logical Table`**. In the Explorer window, a Detail Logical Table is added under the Subject as follows:



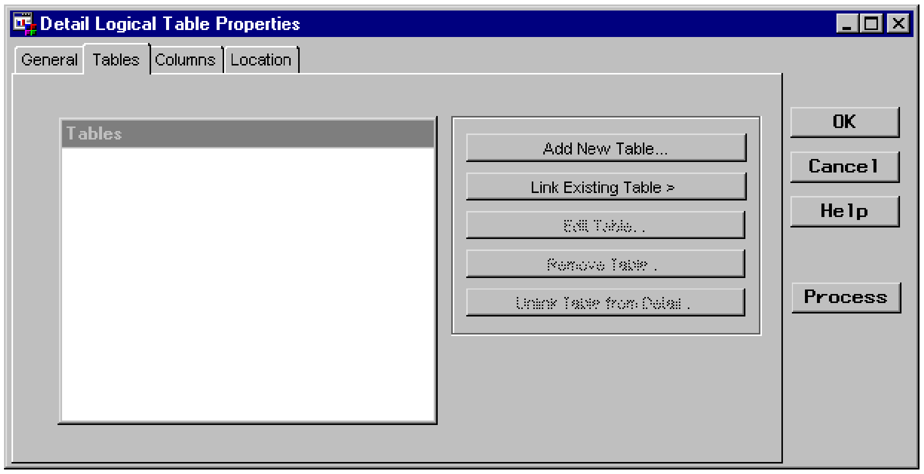
To update the default metadata for the Detail Logical Table, position the cursor on its icon, click the right mouse button, and select **`Properties`**. The Detail Logical Table Properties window displays for you to enter the appropriate information.

General Tab specifies the table name **`Group Detail Logical Table`**, a description, an owner, and an administrator.



Name: Group Detail Logical Table
Description:
Owner: I. Own Whouse
Administrator: Valerie James

Tables Tab specifies the Detail Tables that are members of the Detail Logical Table, which do not exist yet.



The next section describes how to create a Detail Table to add to the Detail Logical Table.

Example: Creating a Detail Table

Overview

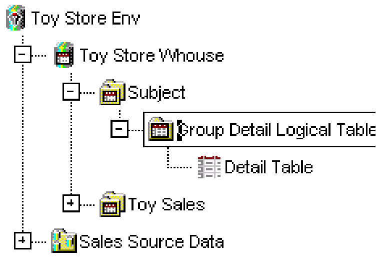
After you have a Detail Logical Table in which you can group Detail Tables, you can add the Detail Tables. There are different methods you can use to add a Detail Table:

- From the Detail Logical Table Properties window, select the Tables tab, and then click Add New Table. The Detail Table Properties window opens for you to define the appropriate metadata for the Detail Table.
- In the SAS/Warehouse Administrator Explorer, position the cursor on the Detail Logical Table icon, click the right mouse button, and select **Add New Table**. The following explanations describe this method.

Note: The following explanations describe the metadata and methods used to achieve the desired results. It is assumed that the appropriate Data Warehouse Environment, Data Warehouse, Subject, Detail Logical Table, and ODD exist. △

Define Detail Table Properties

In the SAS/Warehouse Administrator Explorer, position the cursor on the Detail Logical Table (for example, **Group Detail Logical Table**), click the right mouse button, and select **Add New Table**. In the Explorer window, a new Detail Table is added under the Detail Logical Table as follows:



To update the default metadata for the table, position the cursor on its icon, click the right mouse button, and select **`Properties`**. The Detail Table Properties window displays for you to enter the appropriate information.

General Tab specifies the table name **`Customer`**, a description, an owner, and an administrator.

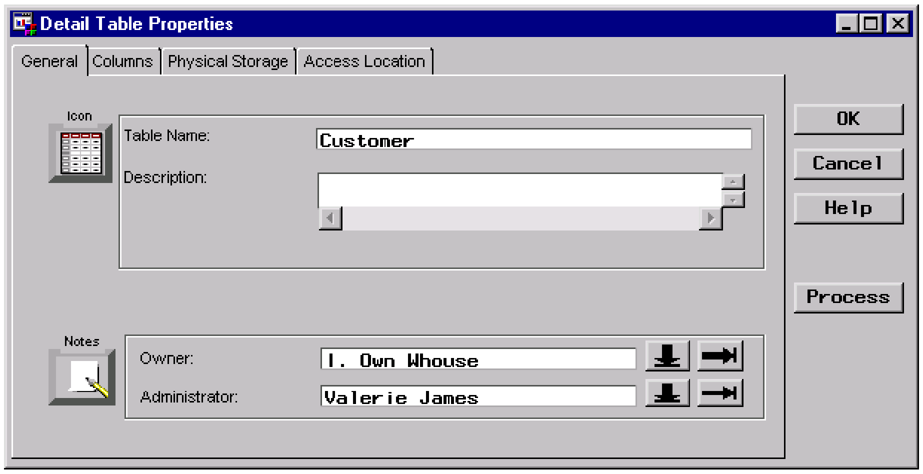
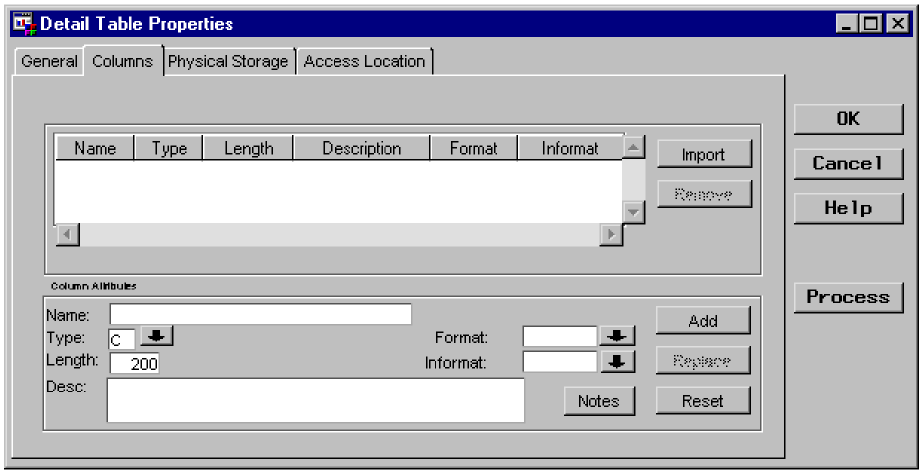


Table Name: Customer
Description:
Owner: I. Own Whouse
Administrator: Valerie James

Columns Tab specifies the columns to be included in the Detail Table, which does not exist yet.

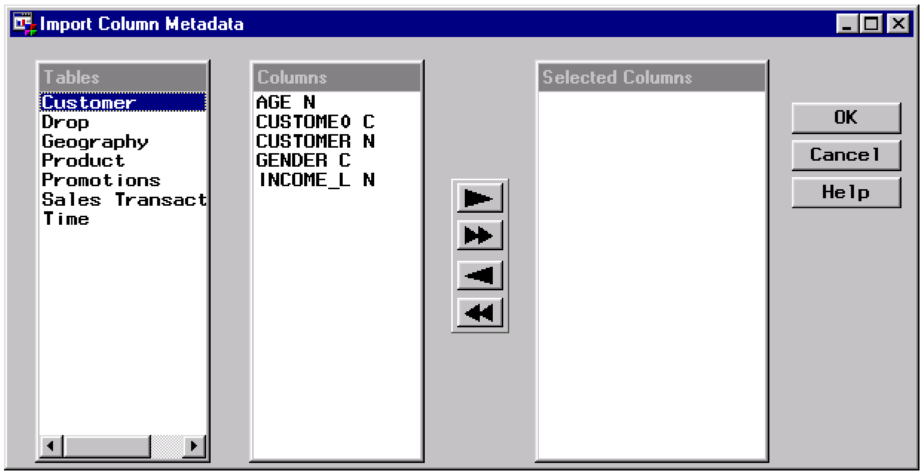


Name	Type	Length	Description	Format	Informat

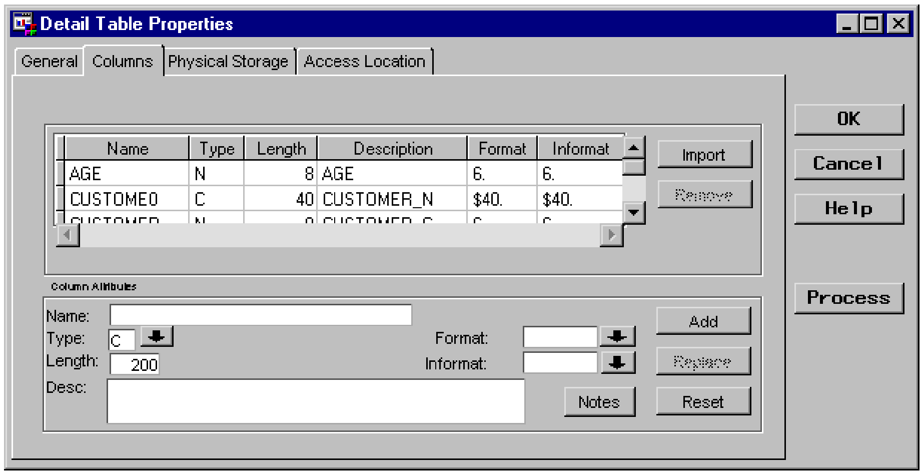
Name:
Type: C
Length: 200
Desc:
Format:
Informat:

To import columns from an input source, click Import to display a list of input sources. Select an input source, for example **`Operational Data Sources`**. The Import Column Metadata window displays.

From the Import Column Metadata window, which lists the available tables, select an ODD, for example **Customer**, to display its available columns.



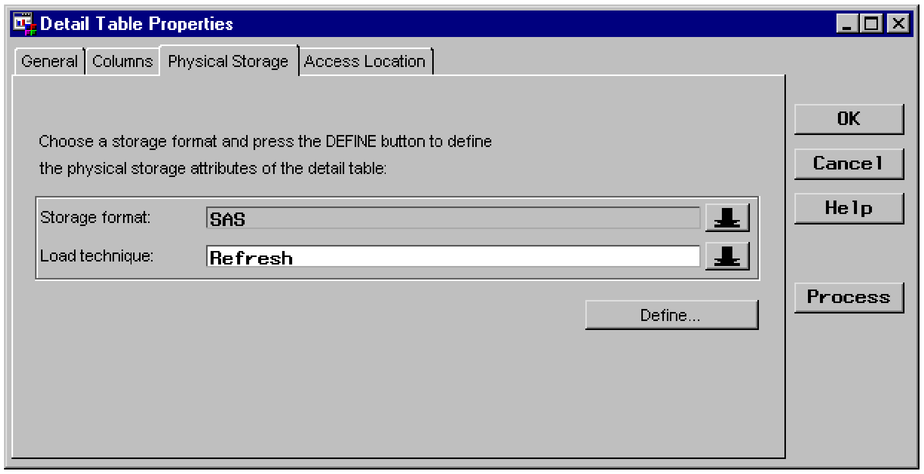
Select the appropriate columns listed under **Columns**, which for this example are all of the columns. Use the double arrows to move these columns to **Selected Columns**, and then click OK. You are returned to the Columns tab in the Detail Table Properties window, which now lists the imported columns.



Name	Type	Length	Description	Format	Informat
AGE	N	8	AGE	6.	6.
CUSTOME0	C	40	CUSTOMER_N	$40.	$40.

Name:
Type: C
Length: 200
Desc:
Format:
Informat:

Physical Storage Tab	specifies physical storage attributes. This example specifies the storage format **SAS** and the load technique **Refresh**.


Detail Table Properties

General | Columns | Physical Storage | Access Location

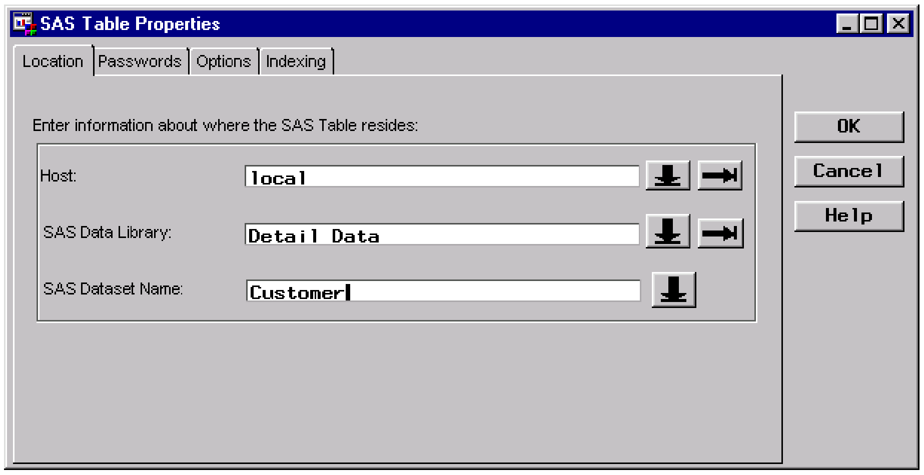
Choose a storage format and press the DEFINE button to define the physical storage attributes of the detail table:

Storage format: SAS

Load technique: Refresh

Click Define to open the SAS Table Properties window and view the table metadata.

Location Tab specifies where the SAS table is stored.


SAS Table Properties

Location | Passwords | Options | Indexing

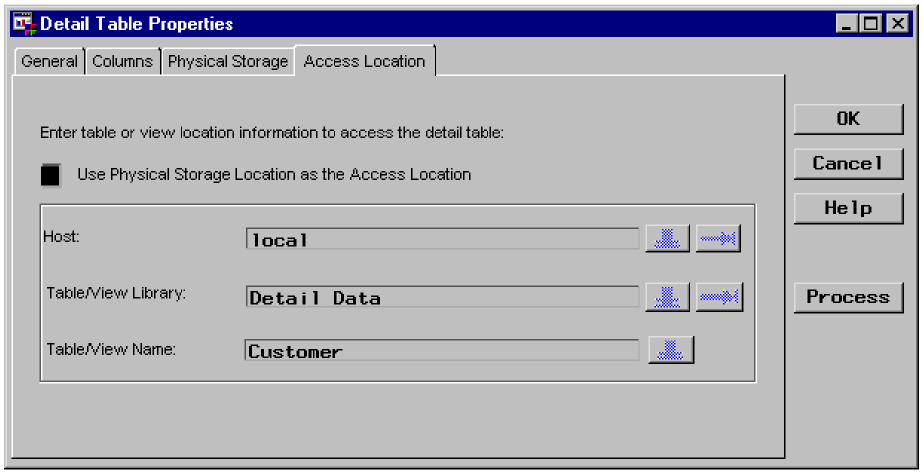
Enter information about where the SAS Table resides:

Host: local

SAS Data Library: Detail Data

SAS Dataset Name: Customer

Access Location Tab specifies a location to provide interactive access to the Detail Table. For this example, the check box **`Use Physical Storage Location as the Access Location`** is selected by default, which enables the software to use information from the Physical Storage tab to provide interactive access to the table.



Enter table or view location information to access the detail table:

Use Physical Storage Location as the Access Location

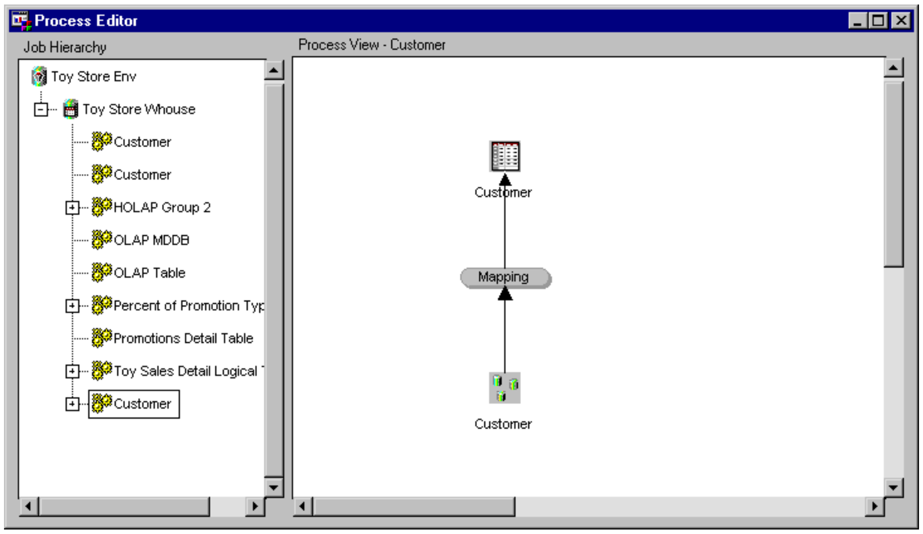
Host: local

Table/View Library: Detail Data

Table/View Name: Customer

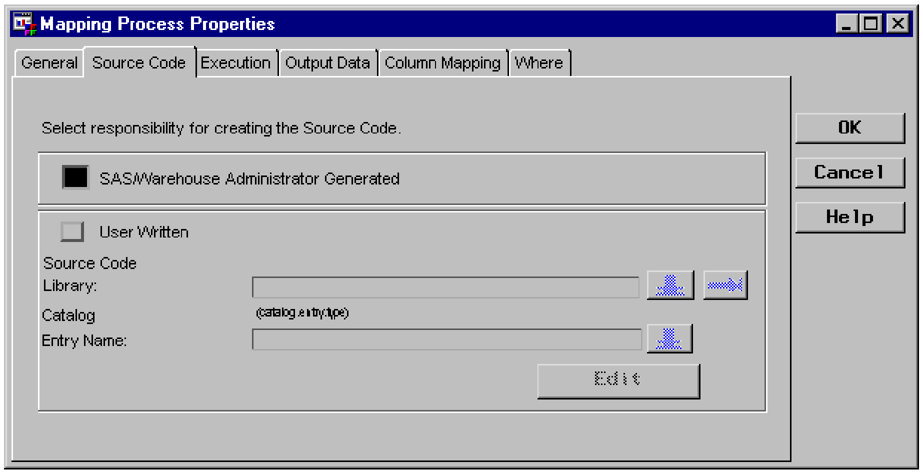
Define Process Editor Job

In the Process Editor Job, the Detail Table **Customer** is specified as the output target and the ODD **Customer** is specified as the input source. The following Process Editor window shows the Process Flow for the Job:



The processes defined in the Job are as follows:

Mapping Process □ The source code to map columns is generated by SAS/Warehouse Administrator, rather than user written, as shown in the Source Code tab:



Mapping Process Properties

General | Source Code | Execution | Output Data | Column Mapping | Where

Select responsibility for creating the Source Code.

■ SAS/Warehouse Administrator Generated

☐ User Written

Source Code Library:

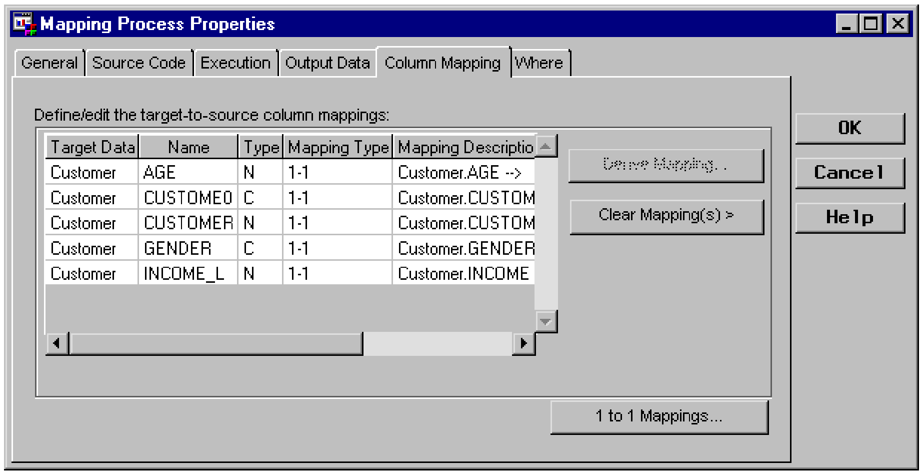
Catalog Entry Name:

(catalog.entry.type)

Edit

OK | Cancel | Help

□ Column mapping is defined as one-to-one mapping, as shown in the Column Mapping tab:



Mapping Process Properties

General | Source Code | Execution | Output Data | Column Mapping | Where

Define/edit the target-to-source column mappings:

Target Data	Name	Type	Mapping Type	Mapping Descriptio
Customer	AGE	N	1-1	Customer.AGE -->
Customer	CUSTOMEO	C	1-1	Customer.CUSTOM
Customer	CUSTOMER	N	1-1	Customer.CUSTOM
Customer	GENDER	C	1-1	Customer.GENDER
Customer	INCOME_L	N	1-1	Customer.INCOME

Derive Mapping...

Clear Mapping(s) >

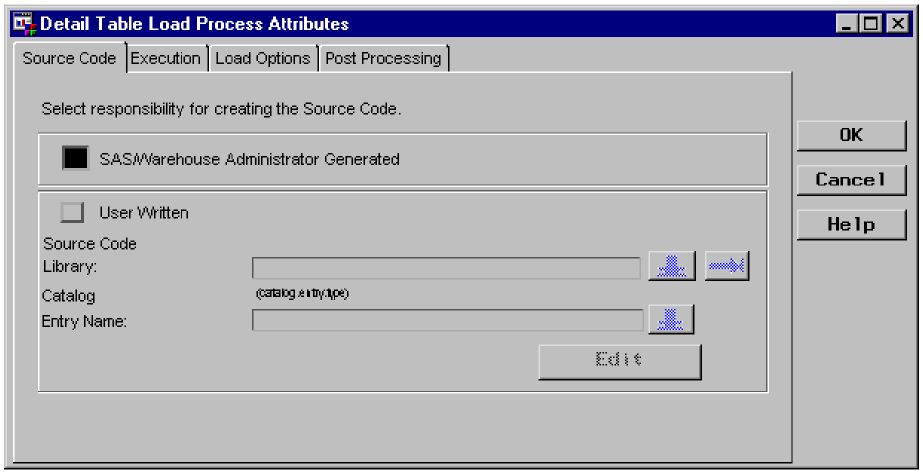
1 to 1 Mappings...

OK | Cancel | Help

Note: To produce one-to-one mapping, first click 1 to 1 Mappings on the Column Mapping tab, which opens the One-to-One Column Mapping window. Then, click Quick Map. △

Load Step Process

The source code is generated by SAS/Warehouse Administrator, rather than user written, as shown in the Detail Table Load Process Attributes window:



Select responsibility for creating the Source Code.

■ SAS/Warehouse Administrator Generated

☐ User Written

Source Code Library:

Catalog Entry Name: (catalog.entry.type)

For more information about Process Editor Jobs, see Chapter 13, "Maintaining Jobs," on page 251. For more information about processes, see Chapter 14, "Maintaining Processes," on page 281.

Example: Linking from a Detail Logical Table to an Existing Detail Table

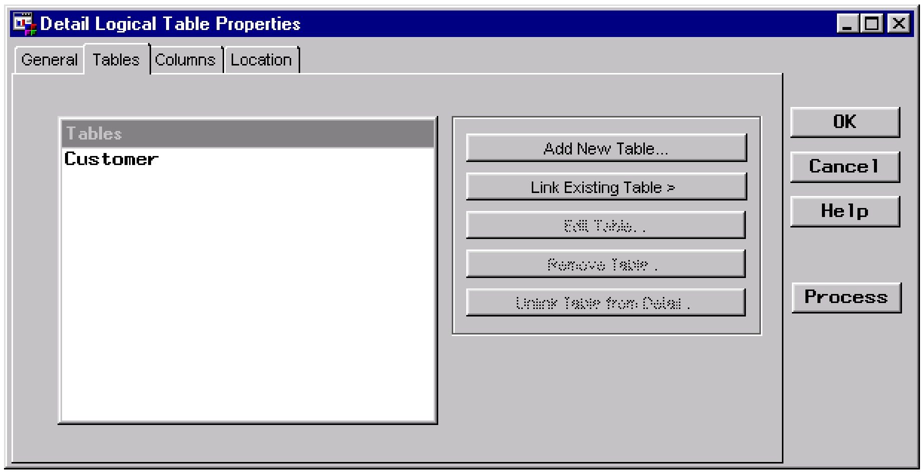
Overview

SAS/Warehouse Administrator provides a linking capability that enables Detail Logical Tables in different Subjects to share individual Detail Tables. That is, a single Detail Table needs to be defined and stored only once but can be linked to from any number of Detail Logical Tables.

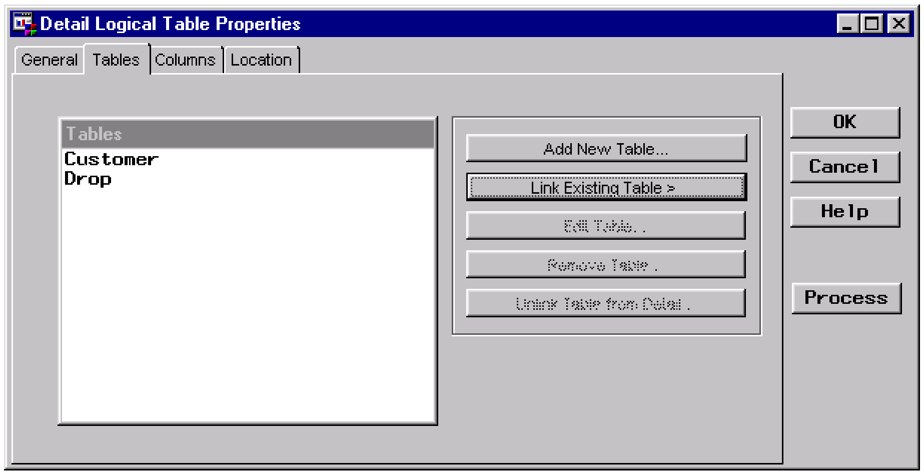
Note: The following explanations describe the metadata and methods used to achieve the desired results. It is assumed that the appropriate Data Warehouse Environment, Data Warehouse, Subjects, Detail Logical Tables, and Detail Table exist. △

Create a Link

To link from a Detail Logical Table to an existing Detail Table (in another Detail Logical Table), position the cursor on the Detail Logical Table icon from which you want to create the link, click the right mouse button, and select **`Properties`**. The Detail Logical Table Properties window displays. Select the Tables tab, which specifies the Detail Tables that are members of the Detail Logical Table.



Click Link Existing Table to display a list of existing Detail Tables, and then select a Detail Table from the list. The selected Detail Table becomes a member (by means of a link) of the current Detail Logical Table.



Example: Creating a Detail Logical Table as a View to Multiple Detail Tables

Overview

A common use of a Detail Logical Table is to create it as a view to multiple, related Detail Tables. The result is that you use the Detail Logical Table as a detail data store, which can then be used as an input source for other SAS/Warehouse objects, such as an OLAP summary data store.

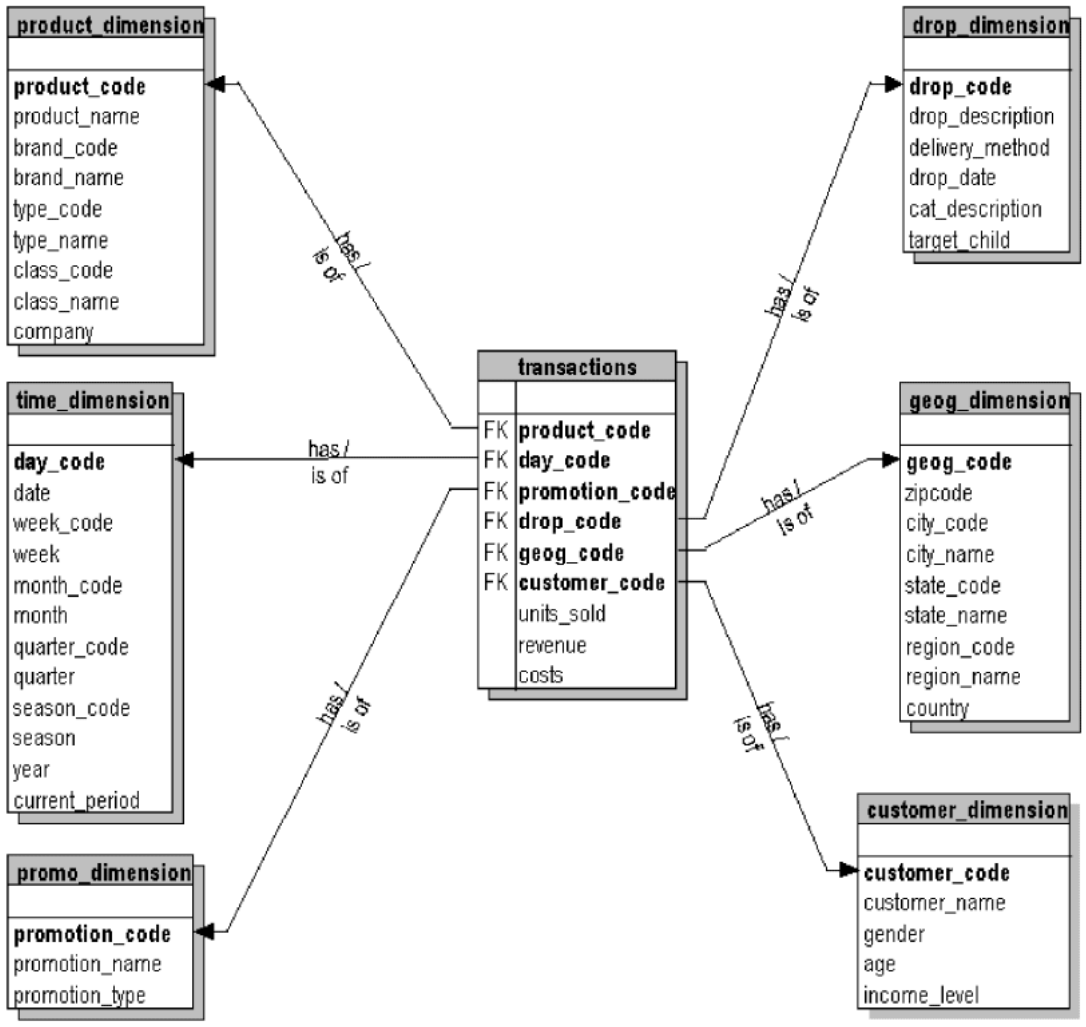
Note: The following explanations describe the metadata and methods used to achieve the desired results. It is assumed that the appropriate Data Warehouse Environment, Data Warehouse, Subject, and ODDs exist. △

Planning the Detail Logical Table to be Organized as a Star Schema

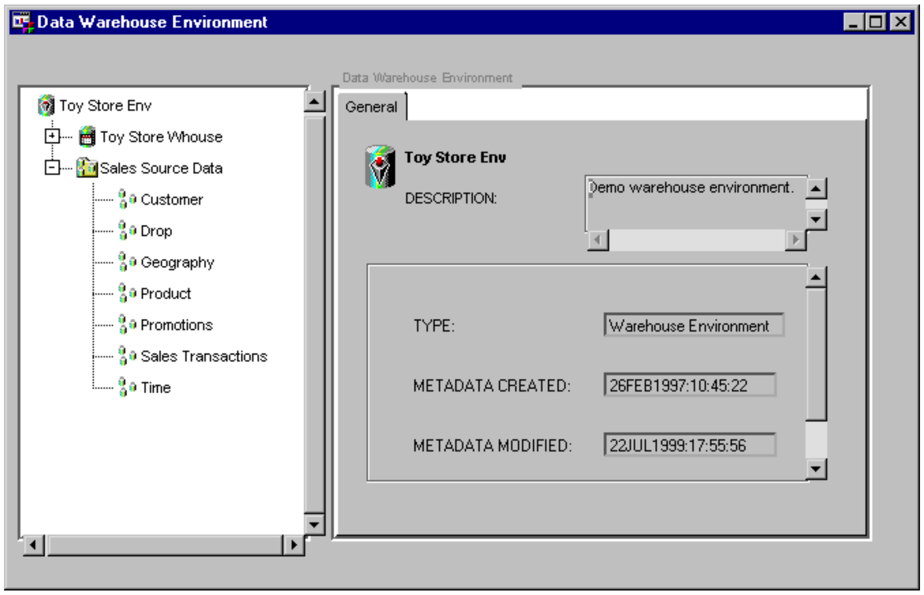
To illustrate how to create a Detail Logical Table as a view to multiple, related Detail Tables, this example creates a Detail Logical Table that organizes data in a *star schema*. A star schema is an arrangement of tables in which a large fact table has a composite, a foreign key, and is joined to several dimension tables.

For this example, the Detail Logical Table uses the toy sales data. The star schema organization is centered around sales transactions, which is joined to several dimension tables, with each dimension table having a single primary key. Figure 10.1 on page 166 shows the star schema organization:

Figure 10.1 Toy Sales Star Schema



Each Detail Table to be viewed by the Detail Logical Table obtains its data from a specific ODD. That is, the following ODDs exist:



Toy Store Env

DESCRIPTION: Demo warehouse environment.

TYPE: Warehouse Environment

METADATA CREATED: 26FEB1997:10:45:22

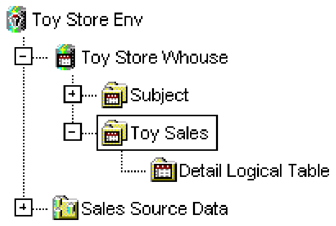
METADATA MODIFIED: 22JUL1999:17:55:56

For this example, the Detail Tables will be defined as follows:

- a fact Detail Table named **`Fact`** with input from ODD **`Sales Transactions`**
- six dimension Detail Tables:
  - Detail Table **`Customer`** with input from ODD **`Customer`**
  - Detail Table **`Drop`** with input from ODD **`Drop`**
  - Detail Table **`Geography`** with input from ODD **`Geography`**
  - Detail Table **`Product`** with input from ODD **`Product`**
  - Detail Table **`Time`** with input from ODD **`Time`**

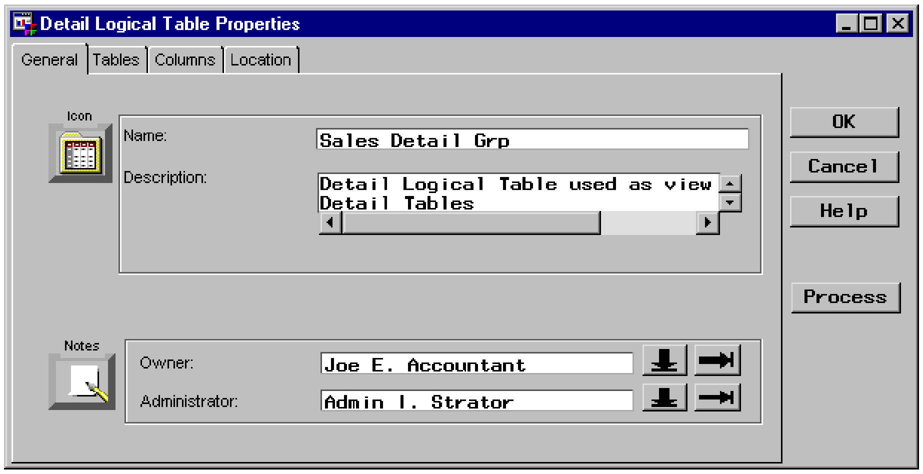
Define Detail Logical Table Properties

In the SAS/Warehouse Administrator Explorer, position the cursor on the Subject, for example **`Toy Sales`**, click the right mouse button, select **`Add Item`**, and then **`Detail Logical Table`**. In the Explorer window, a new Detail Logical Table is added under the Subject as follows:



To update the default metadata for the Detail Logical Table, position the cursor on its icon, click the right mouse button, and select **`Properties`**. The Detail Logical Table Properties window displays for you to enter the appropriate information.

General Tab	specifies the table name **`Sales Detail Grp`**, a description, an owner, and an administrator.


Detail Logical Table Properties

General | Tables | Columns | Location

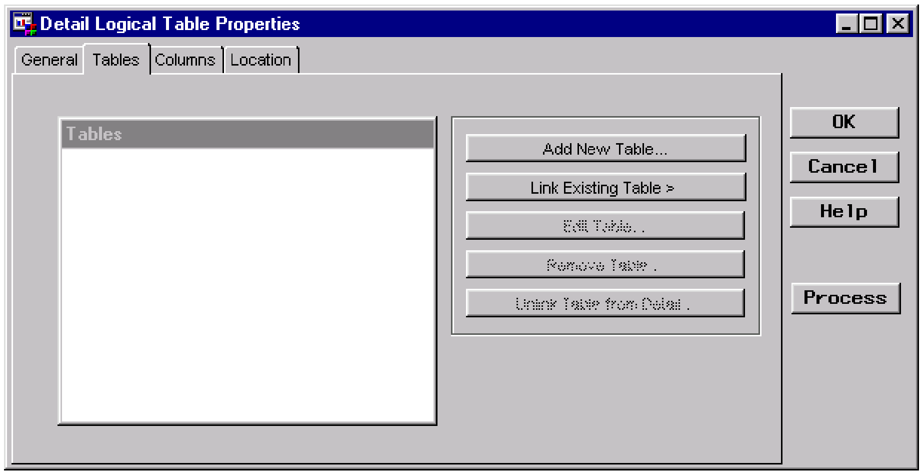
Name: Sales Detail Grp

Description: Detail Logical Table used as view Detail Tables

Owner: Joe E. Accountant

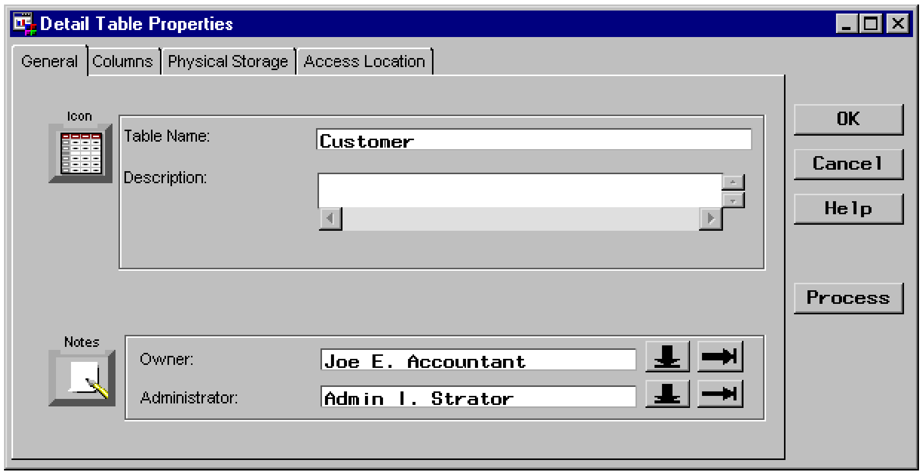
Administrator: Admin I. Strator

Tables Tab specifies the Detail Tables to be members of the Detail Logical Table, which do not exist yet.


Detail Logical Table Properties

General | Tables | Columns | Location

To add a table, click $\boxed{\text{Add New Table}}$ to display the Detail Table Properties window. Enter the name of the Detail Table, for example **`Customer`**.


Detail Table Properties

General | Columns | Physical Storage | Access Location

Table Name: Customer

Description:

Owner: Joe E. Accountant

Administrator: Admin I. Strator

To finish defining the Detail Table, complete the metadata for the remaining Detail Table Properties window tabs. For a description of adding a Detail Table, see "Define Detail Table Properties" on page 158.

After you define the Detail Table, click OK to add the Detail Table to the list of member tables in the Detail Logical Table. You are returned to the Tables tab in the Detail Logical Table Properties window.

You would repeat these steps adding each Detail Table, which for this example, includes the following Detail Tables:

Customer (added)

Drop

Fact

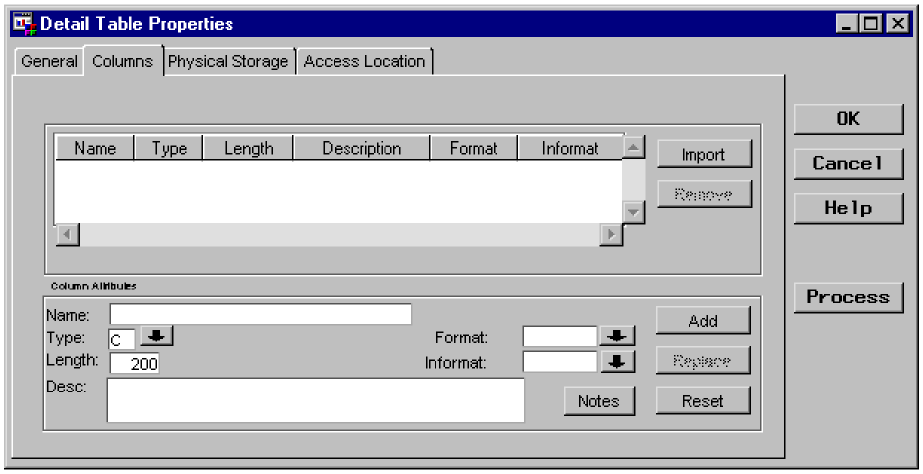
Geography

Product

Promotions

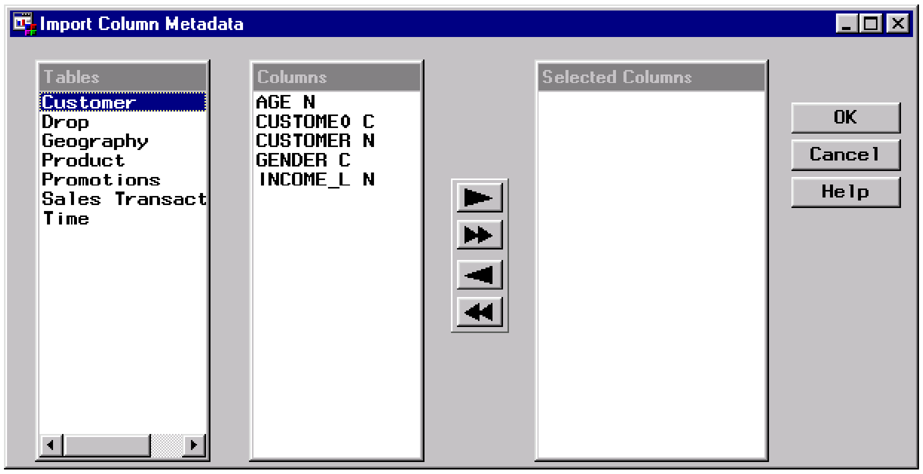
Time

Columns Tab specifies the Detail Table columns to be viewed by the Detail Logical Table, which have not been specified yet.



Name:
Type: C
Length: 200
Desc:
Format:
Informat:

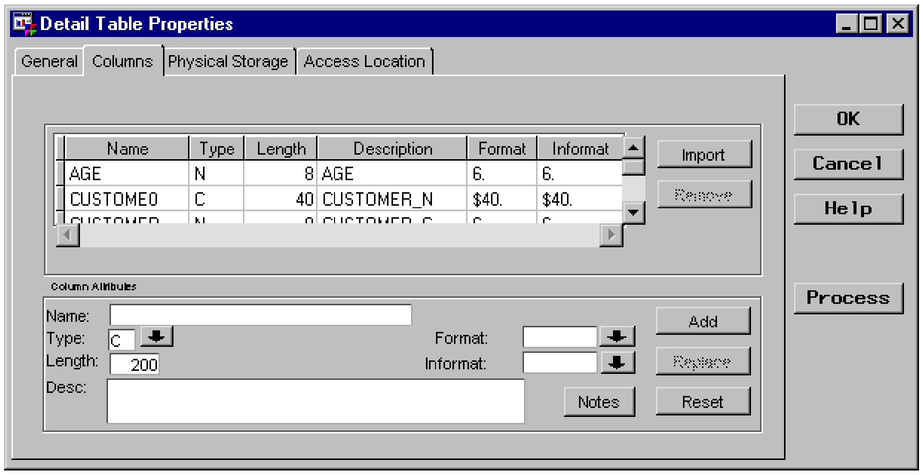
To import columns from an input source, click Import to display a list of input sources, then select **`Detail Tables`**. From the Import Column Metadata window, which lists the available tables, select a table (for example, **`Customer`**) to display its available columns.



Tables:
- Customer
- Drop
- Geography
- Product
- Promotions
- Sales Transact
- Time

Columns:
```
AGE N
CUSTOME0 C
CUSTOMER N
GENDER C
INCOME_L N
```

Select the appropriate columns listed under **`Columns`**, which for this example are all the columns, use the double arrows to move them to **`Selected Columns`**, then click OK. You are returned to the Columns tab in the Detail Logical Table Properties window.



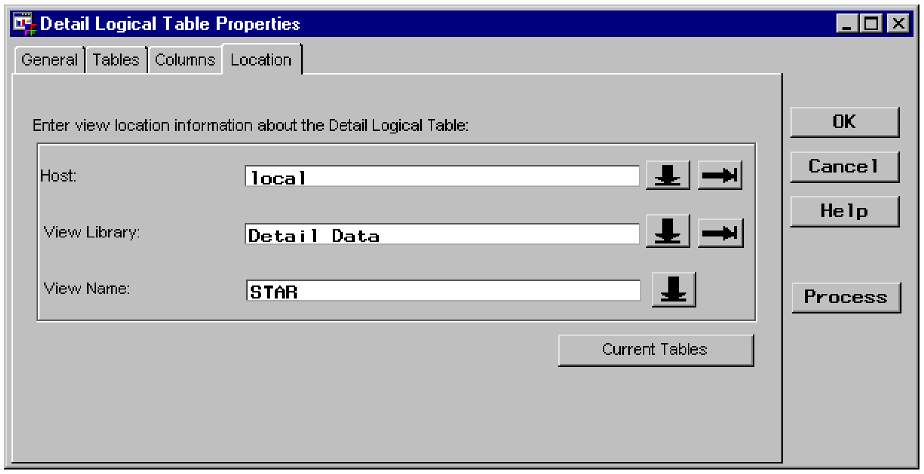
Name	Type	Length	Description	Format	Informat
AGE	N	8	AGE	6.	6.
CUSTOME0	C	40	CUSTOMER_N	$40.	$40.

Name:
Type: C
Length: 200
Desc:
Format:
Informat:

Repeat the column import process for each remaining member Detail Table.

Note: When implementing a star schema, the key column, which links the fact table to each dimension table, is included in both tables. You need only to import that column for the Detail Logical Table from either the fact Detail Table or the dimension Detail Table. Otherwise, you will get a dialog message indicating that a duplicate column exists and asking you to rename it. If you import a key column twice, you can cancel the rename dialog if you choose. △

Location Tab
specifies the physical location of the view that provides interactive access to the Detail Logical Table.



Enter view location information about the Detail Logical Table:

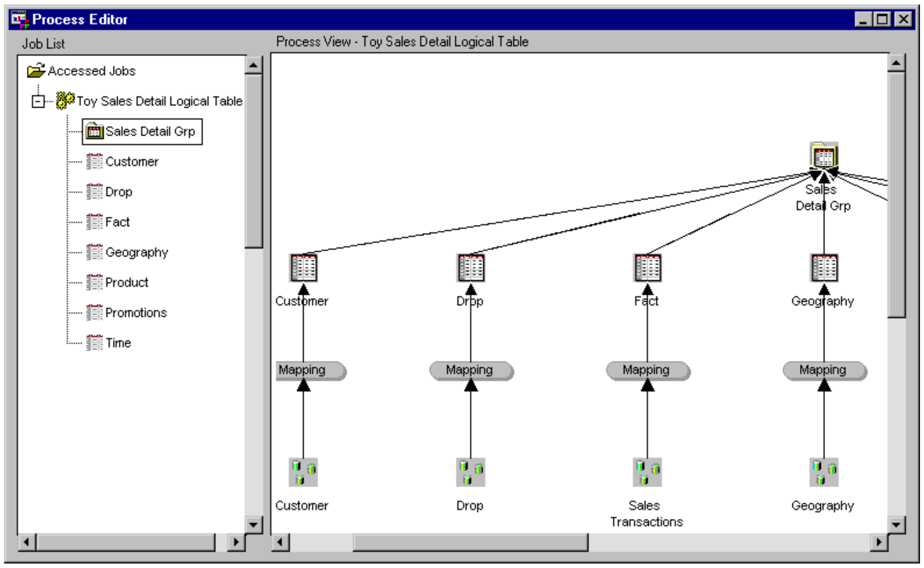
Host: local

View Library: Detail Data

View Name: STAR

Define Process Editor Job

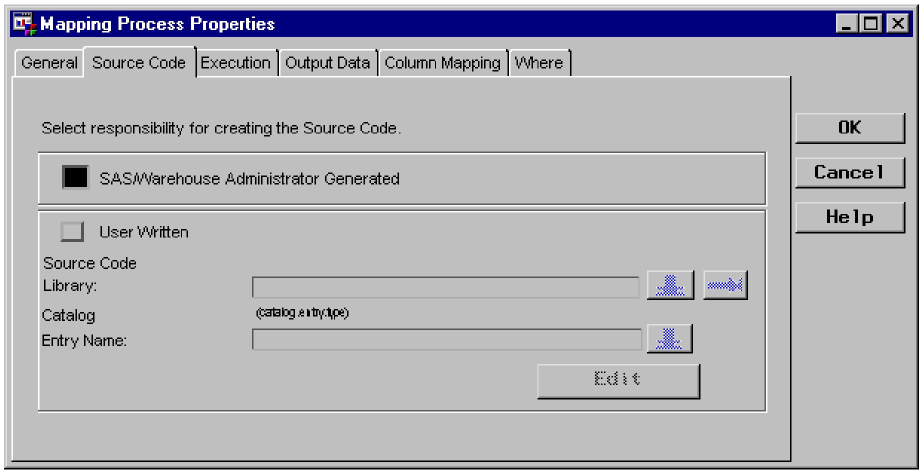
In the Process Editor Job, the Detail Logical Table and all its children Detail Tables are specified in a single Job. **`Sales Detail Grp`** is specified as the output target. Each Detail Table that is a member of the Detail Logical Table is specified as input to the Detail Logical Table as well as an output target in the Job with corresponding ODDs specified as the input sources. The following Process Editor window shows the Process Flow for the Job:



Note: For this example, a single Process Editor Job is illustrated with multiple input sources and output targets. However, you could also define separate Jobs for each Detail Table and a Job for the Detail Logical Table. △

The processes defined in the Job are as follows:

Mapping Processes	The source code to map columns is generated by SAS/Warehouse Administrator, rather than being user written. The Source Code tab is shown for the **`Customer`** Detail Table:



Select responsibility for creating the Source Code.

[✓ SAS/Warehouse Administrator Generated]

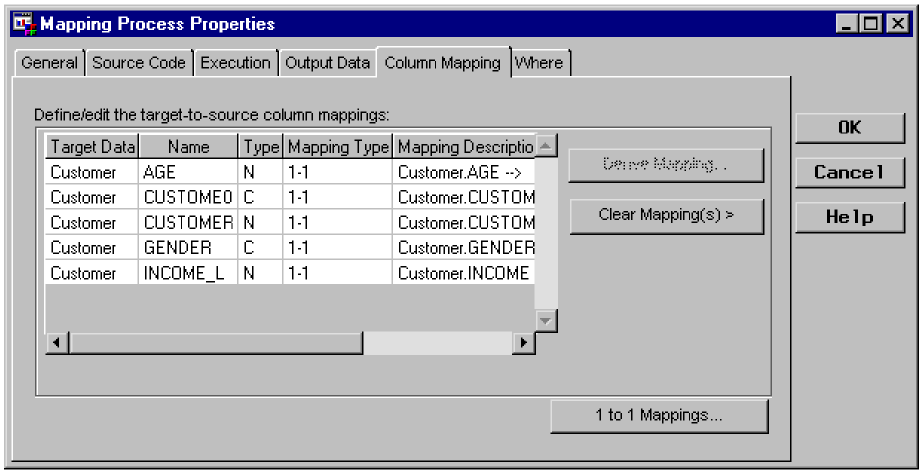
User Written

Source Code Library:

Catalog Entry Name:

(catalog.entry.type)

Column mapping is defined by one-to-one mapping and some data transformations. The Column Mapping tab is shown for the **Customer** Detail Table:



Define/edit the target-to-source column mappings:

Target Data	Name	Type	Mapping Type	Mapping Descriptio
Customer	AGE	N	1-1	Customer.AGE -->
Customer	CUSTOMEO	C	1-1	Customer.CUSTOM
Customer	CUSTOMER	N	1-1	Customer.CUSTOM
Customer	GENDER	C	1-1	Customer.GENDER
Customer	INCOME_L	N	1-1	Customer.INCOME

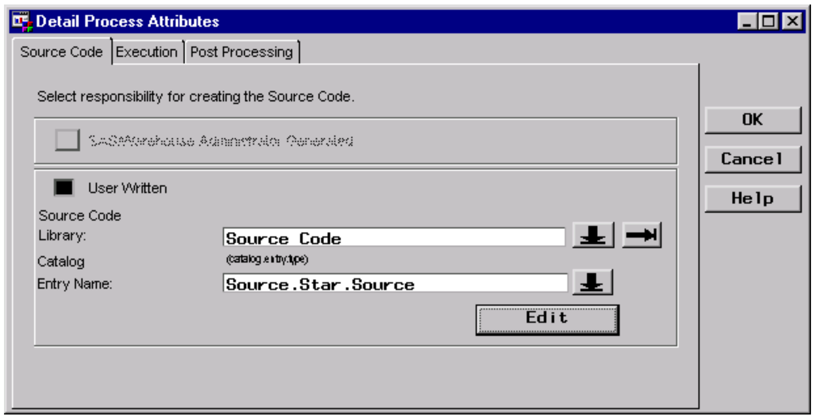
Note: To produce one-to-one mapping, click 1 to 1 Mappings from the Column Mapping tab, which opens the One-to-One Column Mapping window. Then, click Quick Map. △

Note: To define data transformations, first select the column(s) to be transformed, then click Derive Mapping, which opens the Expression Builder window. Use the Expression Builder window to define the transformation. △

Load Step Processes

The source code to load each Detail Table is generated by SAS/Warehouse Administrator.

The source code to load the Detail Logical Table is user written, as shown in the Detail Process Attributes window. For a Detail Logical Table, you must supply the Load Step code. The code creates an SQL view. (Note that you can use the Query window to generate SQL code.)



Select responsibility for creating the Source Code.

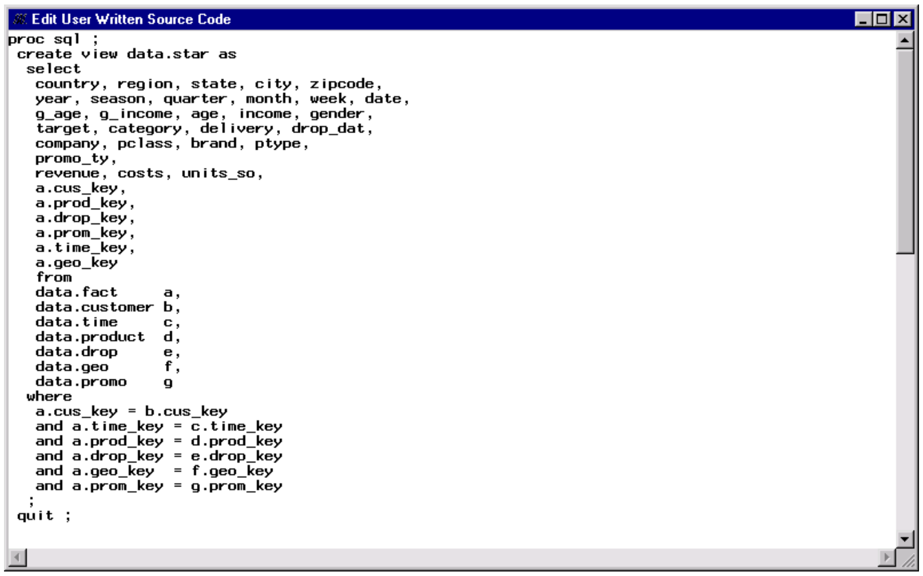
SAS/Warehouse Administrator Generated

■ User Written

Source Code Library: Source Code

Catalog Entry Name: Source.Star.Source (catalog.entry.type)

The code used to create the SQL view follows:



```
proc sql ;
 create view data.star as
  select
   country, region, state, city, zipcode,
   year, season, quarter, month, week, date,
   g_age, g_income, age, income, gender,
   target, category, delivery, drop_dat,
   company, pclass, brand, ptype,
   promo_ty,
   revenue, costs, units_so,
   a.cus_key,
   a.prod_key,
   a.drop_key,
   a.prom_key,
   a.time_key,
   a.geo_key
   from
   data.fact     a,
   data.customer b,
   data.time     c,
   data.product  d,
   data.drop     e,
   data.geo      f,
   data.promo    g
  where
   a.cus_key = b.cus_key
   and a.time_key = c.time_key
   and a.prod_key = d.prod_key
   and a.drop_key = e.drop_key
   and a.geo_key  = f.geo_key
   and a.prom_key = g.prom_key
  ;
 quit ;
```

For more information about Process Editor Jobs, see Chapter 13, "Maintaining Jobs," on page 251. For more information about processes, see Chapter 14, "Maintaining Processes," on page 281.

What's Next

After you create detail data stores, you can use them as inputs to other SAS/Warehouse Administrator objects such as OLAP summary data stores, or you can exploit them with tools designed to work with detail data, such as data mining applications.

CHAPTER

11

Maintaining OLAP Groups and OLAP Summary Data Stores

Overview

OLAP (online analytical processing) is a reporting technology that provides high-performance analysis and easy reporting of large amounts of data, that have been summarized. OLAP reporting functionality is possible because the detail data has been transformed—set up for an OLAP application to reflect the dimensions of the data. That is, the following has occurred:

1. The detail data has been summarized, which is a process that generates derived summary data by applying calculations to input data.
2. The derived summary data is stored in a summary data store, which is a physical storage file such as a relational table or an MDDB (multidimensional database).
3. The summary data store is available to an OLAP application.

SAS/Warehouse Administrator supports OLAP reporting by enabling you to create these data warehouse objects:

OLAP Group	is a grouping element that organizes related summary data, which is stored in OLAP Tables and OLAP MDDBs. OLAP Group properties specify the logical structure of the summarized data. You can add an OLAP Group to a Subject only.
OLAP Table	is a metadata record that specifies a file for storing derived summary data. This file can be a SAS table or view or a DBMS table or view. To load an OLAP Table, SAS/Warehouse Administrator generates code for the SUMMARY procedure, which summarizes data by computing descriptive statistics for columns across all rows or within groups of rows. Note that an OLAP Table can also be used by non-OLAP applications. You can add an OLAP Table to an OLAP Group only.

OLAP MDDB	is a metadata record that specifies a SAS MDDB (multidimensional database). 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. You can add an OLAP MDDB to an OLAP Group only.

In general, to maintain OLAP Groups and OLAP summary data stores, you will do the following:

1 Analyze the detail data to understand its content and the relationships among the data.
2 Evaluate the most likely data access patterns to determine the storage strategy.
3 Define the properties of the OLAP Group.
4 Define the properties of the summary data store(s) for the OLAP Group; that is, define the OLAP Tables and OLAP MDDBs.
5 Define the Process Editor Job(s) that include the data preparation processes, which prepare the data to be summarized, and the Load Steps, which define the steps to summarize the data and load the OLAP summary data stores with derived data.
6 Execute the Jobs.
7 Verify that the summary data stores are loaded; that is, check logs, use the data utilities, and so on.

After you have defined and loaded OLAP objects, the OLAP metadata can be used by an OLAP application. Using SAS/Warehouse Administrator, for example, you can export the OLAP metadata to SAS/EIS software.

Note: The basic steps for creating OLAP objects are described in the online Help. To display the relevant online Help, in the SAS System Help contents, select **`Help on SAS Software Products`**, then **`Using SAS/Warehouse Administrator Software`**. Finally, select **`Defining Summary Data Stores`**. In addition, each SAS/Warehouse Administrator window has associated reference information for its fields and buttons by selecting the Help button. △

Preparing to Create Summary Data Stores

Before you create an OLAP Group and its associated summary data stores, you must do some preparation, which is summarized as follows:

Hierarchy of objects	□ In SAS/Warehouse Administrator, make sure that you have created the appropriate Data Warehouse Environment, Data Warehouse, and Subject.
Input source(s)	□ Determine which input source(s) to use for each summary data store. For example, if the input source for an OLAP MDDB is a Detail Table, the Detail Table must be fully operational before the OLAP MDDB is populated.
	□ Analyze the input data. For details, see “Analyzing Detail Data for an OLAP Application” on page 178.
	□ Decide what columns are needed for the summary data store.

Processes	□ Determine whether the source code, which defines how data moves from input sources to output targets, is generated by SAS/Warehouse Administrator or is it user written and stored in a SAS catalog entry.
	□ Determine the type of Mapping processes, for example, you might need to transform data or generate data.
	□ Determine whether you need to define other processes such as a User Exit process, Data Transfer process, or Record selector process.
Output target	□ Determine how you want to store each summary data store. For example, determine what format you want to store the data, such as SAS format or a DBMS. For more information, see "Summarizing Data Using SAS/Warehouse Administrator" on page 182.
	□ Determine on what platform you want to store the summary data store and where the summarization should occur. That is, you could store the data store locally and have the summarization occur on a remote host.

Analyzing Detail Data for an OLAP Application

Understanding Detail Data

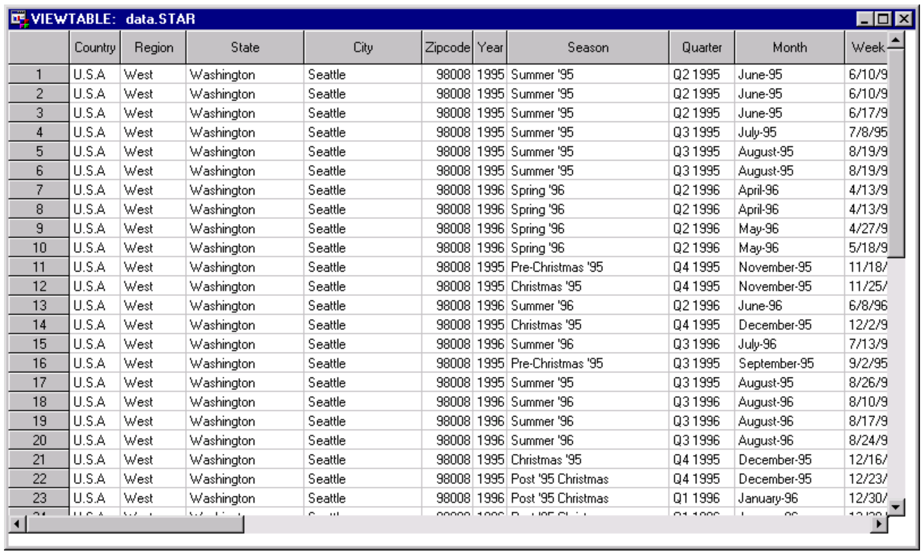
Detail data is information that is at or near the fact level in a database. It is the data that an OLAP application intends to analyze. In SAS/Warehouse Administrator, detail data is stored in

- □ Data Tables
- □ Detail Logical Tables
- □ Detail Tables.

To understand what detail data consists of, consider an OLAP application to analyze the sales of toys for which the detail data represents toy sales transactions across several product lines and manufacturers. For the Toy Store Warehouse, a Subject named **`Toy Sales`** contains the Detail Logical Table named **`Sales Detail Grp`**, which is physically stored as a SAS SQL view named **`DATA.STAR`**. Display 11.1 on page 179 shows some of the detail data in the Detail Logical Table.

Note: For an explanation of how the Detail Logical Table is created and a complete description of its contents, see "Example: Creating a Detail Logical Table as a View to Multiple Detail Tables" on page 165. △

Display 11.1 Sales Detail Grp Detail Logical Table


VIEWTABLE: data.STAR

	Country	Region	State	City	Zipcode	Year	Season	Quarter	Month	Week
1	U.S.A	West	Washington	Seattle	98008	1995	Summer '95	Q2 1995	June-95	6/10/9
2	U.S.A	West	Washington	Seattle	98008	1995	Summer '95	Q2 1995	June-95	6/10/9
3	U.S.A	West	Washington	Seattle	98008	1995	Summer '95	Q2 1995	June-95	6/17/9
4	U.S.A	West	Washington	Seattle	98008	1995	Summer '95	Q3 1995	July-95	7/8/95
5	U.S.A	West	Washington	Seattle	98008	1995	Summer '95	Q3 1995	August-95	8/19/9
6	U.S.A	West	Washington	Seattle	98008	1995	Summer '95	Q3 1995	August-95	8/19/9
7	U.S.A	West	Washington	Seattle	98008	1996	Spring '96	Q2 1996	April-96	4/13/9
8	U.S.A	West	Washington	Seattle	98008	1996	Spring '96	Q2 1996	April-96	4/13/9
9	U.S.A	West	Washington	Seattle	98008	1996	Spring '96	Q2 1996	May-96	4/27/9
10	U.S.A	West	Washington	Seattle	98008	1996	Spring '96	Q2 1996	May-96	5/18/9
11	U.S.A	West	Washington	Seattle	98008	1995	Pre-Christmas '95	Q4 1995	November-95	11/18/
12	U.S.A	West	Washington	Seattle	98008	1995	Christmas '95	Q4 1995	November-95	11/25/
13	U.S.A	West	Washington	Seattle	98008	1996	Summer '96	Q2 1996	June-96	6/8/96
14	U.S.A	West	Washington	Seattle	98008	1995	Christmas '95	Q4 1995	December-95	12/2/9
15	U.S.A	West	Washington	Seattle	98008	1996	Summer '96	Q3 1996	July-96	7/13/9
16	U.S.A	West	Washington	Seattle	98008	1995	Pre-Christmas '95	Q3 1995	September-95	9/2/95
17	U.S.A	West	Washington	Seattle	98008	1995	Summer '95	Q3 1995	August-95	8/26/9
18	U.S.A	West	Washington	Seattle	98008	1996	Summer '96	Q3 1996	August-96	8/10/9
19	U.S.A	West	Washington	Seattle	98008	1996	Summer '96	Q3 1996	August-96	8/17/9
20	U.S.A	West	Washington	Seattle	98008	1996	Summer '96	Q3 1996	August-96	8/24/9
21	U.S.A	West	Washington	Seattle	98008	1995	Christmas '95	Q4 1995	December-95	12/16/
22	U.S.A	West	Washington	Seattle	98008	1995	Post '95 Christmas	Q4 1995	December-95	12/23/
23	U.S.A	West	Washington	Seattle	98008	1996	Post '95 Christmas	Q1 1996	January-96	12/30/

To understand the detail data, consider the following:

- □ As a whole, the toy sales transactions form a *population*. For each transaction, specific information is captured about the context of that transaction including the time and location of the transaction. Information about the participants in the transaction is also part of the context, such as age, gender, and income of the purchaser, and promotional campaigns implemented by the vendor.
- □ All of that information can be divided into subpopulations or classifications. The fields used to record that information are referred to as classification columns or *class columns*.

  To be useful, the class columns need to break the population into a manageably small number of subpopulations. In order to make selection possible, several class columns with different granularity are often stored. For example, geographic classifications can be done at the Country, Region, State, City, and Zipcode level. The number of discrete values for any class column within the population is referred to as its *cardinality*.

  The idea of OLAP is to explore a large population looking for trends in the measurements within or between many different subpopulations. There can be many different subpopulations defined by combinations of values of the class columns.
- □ In addition to class columns, there are certain properties of a transaction that are measured and stored numerically. These are the measurements, which for the toy sales data are the number of toys sold, the cost to the seller, and the price paid by the buyer. The fields that store the raw data are referred to as *analysis columns* because they are what you want to analyze using OLAP techniques. When you combine the measurements across a number of individuals in a subpopulation, one or more *statistic columns* can be produced per analysis column. That is, the values for an analysis column are used to compute the output summary statistics, which become the values for the statistic column(s) in the summary data store. Typical statistics are sum, minimum, and maximum.
- □ It is possible to store other information about a transaction that is neither classification nor measurement. These are referred to as identification columns or *ID columns*, because they are often used to further identify groups of transactions

that are otherwise identified by class column values. For example, along with the class column **`state_code`** you could have the ID column **`state_name`**.

Determining Dimensions in the Data

To perform OLAP reporting, the detail data must be summarized into multidimensional views, which is a way of looking at data that is organized into dimensions. A dimension acts as an index by which you can access facts according to the value (or values) you want. A dimension is a logical grouping of related columns.

For example, continuing with the **`Sales Detail Grp`** detail data, you could organize the data into these dimensions by grouping class columns based on related characteristics:

Time Dimension
Year, Season, Quarter, Month, Week, Date

Promotion Dimension
Promotion Type, Promotion

Drop Dimension
Target, Category, Delivery, Drop Date, Drop Description

CustomerIncome Dimension
Income Group, Income, Customer Name

CustomerAge Dimension
Age Group, Age, Customer Name

CustomerGender Dimension
Gender, Customer Name

Product Dimension
Company, Class, Brand, Type, Product

Geography Dimension
Country, Region, State, City, Zipcode

Determining Hierarchical Relationships within Dimensions

After you have class columns grouped into dimensions, you should then determine whether there are hierarchical relationships among the class columns within each dimension. That is, if the class columns within a dimension have a hierarchical relationship, each hierarchy would provide a navigational path in order to drill down (or up) to increasing (or decreasing) levels of detail.

For example, for the Detail Logical Table **`Sales Detail Grp`**, assume that the OLAP application intends to explore the Time dimension (which groups class columns **`year`**, **`season`**, **`quarter`**, **`month`**, **`week`**, and **`date`**) in multiple ways. Therefore, three separate hierarchies can be defined as follows:

TimeWeek Hierarchy
Year, Week, Date

TimeQuarter Hierarchy
Year, Quarter, Month, Date

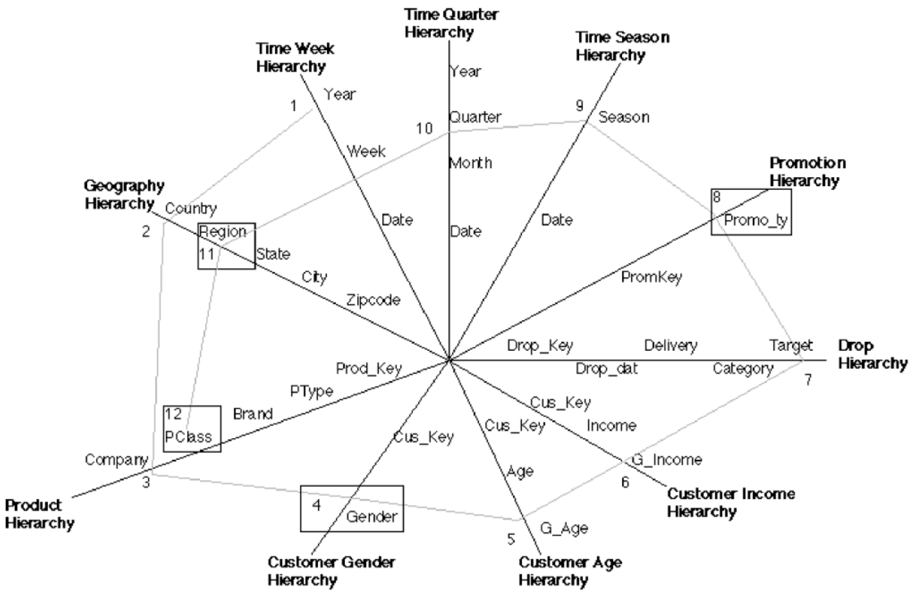
TimeSeason Hierarchy
Season, Date

Determining Crossing(s)

A *crossing* is a unique list of one 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. A crossing represents the physically stored data, which provides the quick response when displaying a report in an OLAP application. For each crossing, there is a single record for each unique combination of values of all named class columns in the original input detail data (population).

There are several methods you can use to help determine crossings. For example, to create a starting point for defining crossings, you can model the data using a *spiral diagram*. Using the toy sales hierarchies, the steps below create the spiral diagram shown in Figure 11.1 on page 181:

Figure 11.1 Spiral Diagram for Toy Sales Detail Data



1 First, draw an axis for each hierarchy. Then, place the class columns on the appropriate axes (working from the outside to the center) in ascending order of cardinality (number of unique values).

Note: The placement of axes in relation to each other can be significant. Try several arrangements to find one that works best. For example, you could arrange the axes in descending order of likelihood of use, in descending order of cardinality at the top dimensional level, or in descending order of number of levels. △

2 Draw a spiral on the diagram to indicate a general ordering scheme for the columns. Start on the outside with the class column with the lowest cardinality, which is also the one most likely to be of interest to OLAP users, and draw a line from this column to an adjacent column. Continue spiraling in toward the center, as shown in Figure 11.1 on page 181.

3 Looking at the spiral diagram, you can produce an ordered list of class columns. Most likely, the more important class columns with the lowest cardinality are at the top and the most detailed class columns with high cardinality are at the bottom. For this example, the order would be:

Year
Country
Company
Gender
G_Age
G_Income
Target
Promo_ty
Season
Quarter
Region
PClass

4 From this list, you can develop a reasonable initial choice of crossings. Start with the entire list and successively drop the highest cardinality column in a "stair-step" fashion to form additional crossings as follows:

```
Year Country Company Gender G_Age G_Income Target Promo_Ty Season Quarter Region PClass
Year Country Company Gender G_Age G_Income Target Promo_Ty Season Quarter Region
Year Country Company Gender G_Age G_Income Target Promo_Ty Season Quarter
Year Country Company Gender G_Age G_Income Target Promo_Ty Season
Year Country Company Gender G_Age G_Income Target Promo_Ty
Year Country Company Gender G_Age G_Income Target
Year Country Company Gender G_Age G_Income
Year Country Company Gender G_Age
Year Country Company Gender
Year Country Company
Year Country
Year
```

Summarizing Data Using SAS/Warehouse Administrator

Generating the Appropriate OLAP Summary Data

The type and number of summary data stores you create depends largely on the data access patterns. That is, there are a wide variety of strategies that you can use ranging from a single OLAP Table with one crossing, or a single OLAP MDDB with multiple crossings, to a proxy MDDB with several associated OLAP MDDBs and OLAP Tables, some of which might reside on an external DBMS.

To facilitate the different strategies required for OLAP reporting, an OLAP Group can be one of the following types, which depends on whether you intend to store summary data in OLAP Tables, OLAP MDDBs, or both:

HOLAP
: supports a hybrid OLAP solution that combines the best features of both ROLAP and MOLAP. HOLAP provides access to diverse data sources on local and remote servers. An OLAP Group of type HOLAP groups both OLAP Tables and OLAP MDDBs, which together represent the data for one OLAP application.

When an OLAP Group of type HOLAP is specified as an output data store in a Process Editor Job, SAS/Warehouse Administrator generates a *proxy MDDB*, which is a physical file that represents the

	structure of the data in an OLAP Group. The proxy MDDB can be used by SAS/EIS software to provide more efficient access to multiple OLAP Tables and OLAP MDDBs.
MOLAP	supports OLAP performed on a multidimensional database, such as a SAS MDDB. SAS/Warehouse Administrator supports MOLAP with an OLAP Group of type MOLAP. Such a group is a grouping mechanism intended to contain only OLAP MDDBs. Multiple MDDBs can be contained in the group, but each MDDB generally represents the data for separate OLAP applications.
ROLAP	supports OLAP performed on a relational database, such as a SAS table or a DBMS table. SAS/Warehouse Administrator supports ROLAP with an OLAP Group of type ROLAP. Such a group is a grouping mechanism intended to contain only OLAP Tables. Multiple OLAP Tables can be contained in the group, but each table generally represents the data for separate OLAP applications. *Note:* SAS/EIS software does not support ROLAP. △
MIXED	groups both OLAP Tables and OLAP MDDBs. Unlike HOLAP, the summary data stores in a MIXED group do not have to be used together. For example, you might choose the MIXED type if you do not want to define several OLAP Groups, with each having only one OLAP Table or OLAP MDDB.

Choosing the Appropriate Data Store

In SAS/Warehouse Administrator, summary data is stored in OLAP Tables and OLAP MDDBs. These are the physical storage units that will contain the derived values for crossings and statistic columns. Here are some considerations when you are determining the type of OLAP summary data store to create:

- If the amount of summary data that you want to store in one data store is very large, consider using an OLAP Table. The amount of data that is considered very large will depend on the storage and processing resources available at your site. The amount could be anywhere from 500 megabytes to 2 gigabytes.
- If you have a specific tool that can access only relational DBMS tables, and you want to use this tool to access your summary data, store that data in an OLAP Table that is a DBMS table.
- To create an OLAP MDDB, SAS/MDDB Server software must be licensed on the machine where the OLAP MDDB will be stored.
- SAS MDDBs use less storage space than SAS or DBMS tables.
- To access data stored in an OLAP MDDB, use the MDDB viewer or a multidimensional SAS/EIS or Web EIS application. (Open the DIR window and type **S** or **B** in front of the MDDB name to view the MDDB's header information or data.) If you want to use other SAS software features to access the data in your summary tables, then use OLAP Tables to store the data.
- Existing records in an MDDB can be updated. However, if you add or drop any statistics or analysis columns (if the metadata definition of the table changes), then the table must be refreshed (rebuilt). If the table is not refreshed, the changes will be ignored.

Assigning OLAP Summary Roles and Defining OLAP Structure

In addition to defining the physical properties of the OLAP Group and the summary data store, you must also

- □ assign the OLAP summary roles, which determine how the columns are used in the summarization process. The summary roles are class columns, statistic columns, and ID columns.
- □ specify the logical structure of the data, which is how the data is to be used by an OLAP report. The structure definitions are the OLAP Cube, dimensions, and hierarchies.

The summary roles and structure definitions include the following:

class column
: is an OLAP summary role that is a numeric or character column used to group data into subpopulations. The values for each class column define groups for analysis. That is, the rows in the detail data store are grouped according to the values of the column, and a separate analysis is run for each group. Class columns typically have a relatively small number of discrete values that define the classification levels of the column.

For example, if columns **`state`** and **`county`** are class columns, you can order the columns so that states come first, and SAS/Warehouse Administrator will summarize data for each county within each state.

Each class column has an associated sort order. The following sort orders are supported:

ASCENDING	for OLAP Tables and OLAP MDDBs, sorts in ascending order by unformatted value. This is the default.
ASCFORMATTED	for OLAP Tables and OLAP MDDBs, sorts in ascending order by formatted value.
DESCENDING	for OLAP MDDBs, sorts in descending order by unformatted value.
DESFORMATTED	for OLAP MDDBs, sorts in descending order by formatted value.
DSORDER	for OLAP Tables and OLAP MDDBs, sorts in the order that the values occur in the input source.

statistic column
: is an OLAP summary role that is a numeric column for storing computed summary statistics, which are the results of the analysis. Values for an input column (analysis column) are used to compute the output summary statistic, which then become the values for the statistic column in the summary data store.

For example, you could add a column named **`minsales`**, assign it as a statistic column using the MIN statistic, then define a Mapping process to compute the derived statistic from an analysis column like **`sales`** to the statistic column **`minsales`**.

Each statistic column has a specific keyword associated with it that specifies which statistic to compute. The following statistics are supported:

SUM	is the sum of nonmissing values for the column. This is the default.
MIN	is the smallest value for the column.
MAX	is the largest value for the column.

N	is the number of rows for the column having nonmissing values.
NMISS	is the number of rows in the column having missing values.
USS	is the uncorrected sum of squares.

ID column
: is an OLAP summary role to include additional columns in the summary data store. You can specify ID columns to an OLAP Table only; it is not supported for an OLAP MDDB.

OLAP Cube
: represents the logical relationships (dimensions and hierarchies) of the OLAP data so that you can run an OLAP report.

For HOLAP, the cube is associated with the OLAP Group and is registered in SAS/EIS software as associated with the proxy MDDB. The result is one OLAP Cube describing the relationships of all the class columns. For MOLAP and ROLAP, there is normally one cube associated with each OLAP MDDB and OLAP Table.

Note: You can decide not to have an OLAP Cube; however, in your OLAP application, you will not be able to drill down without manual intervention. If your intention is to report directly from OLAP data using an OLAP application, then you should define an OLAP Cube. △

dimension
: groups related class columns, which are organized as hierarchies. For example, you could organize sales data into dimensions Geography, Time, and Product. The Time dimension could include multiple hierarchies, such as Time-by-Week and Time-by-Month, which provide different paths in order to drill down to increasing levels of detail.

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. For example, for a dimension named Time, you could define a hierarchy named Time-by-Month that consists of the class columns **`year`**, **`month`**, and **`date`**.

Note: The term hierarchy as used here is not the same as a stored summary level, which is a crossing. △

crossing
: is a unique list of one 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.

A crossing represents the physically stored data, which provides the quick response when displaying a report in an OLAP application. For each crossing, there is a single record for each unique combination of values of all named class columns in the original raw input data. Note that too many crossings result in lots of initial summarization time and disk space for storage, while too few crossings result in slower processing and reporting time for end users.

You can define an NWAY crossing, which is the most detailed type of crossing. An NWAY crossing consists of all the assigned class columns.

Note the following:

- An OLAP MDDB must have an NWAY crossing, and it must be named **`NWAY`** if you intend on using the SAS/Warehouse Administrator code generator.
- For an OLAP Table, you can define a crossing and not specify any class columns in it. The result is summary statistics across all rows in the input source.

When the Process Editor Job for an OLAP object (OLAP Group, OLAP Table, or OLAP MDDB) is executed, SAS/Warehouse Administrator loads the summary data store with the derived, summarized data. As each crossing is accumulated, all records of the input data are sorted into groups by the values of the class columns specified for the crossing. Each group represents a specific subpopulation. Within each group, the summary statistics are derived from the analysis columns in the input data and stored in the summary data store statistic columns. A single record is written to the crossing for each subpopulation.

Example: Creating Summary Data for a HOLAP Application

Overview

HOLAP provides access to diverse data sources. An OLAP Group of type HOLAP groups both OLAP Tables and OLAP MDDBs, which together represent the data for one OLAP application.

Using the **`Sales Detail Grp`** Detail Logical Table discussed in this chapter, consider the design of an OLAP application that calls for storage strategy of two summary data stores:

- one summary table to store the largest crossing, which has twelve class columns
- one SAS MDDB to store the other eleven crossings.

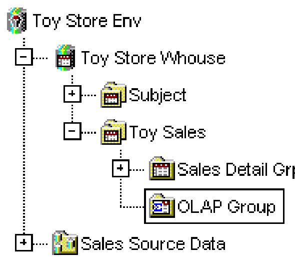
To produce the appropriate summary data for the HOLAP application:

1. Create an OLAP Group of type HOLAP in which you import all of the columns from the detail data source, assign the OLAP summary roles (class and statistic columns), and define structure definitions (OLAP Cube, dimensions, and hierarchies).
2. Create one OLAP Table in which you import a subset of columns and OLAP summary roles from the group and define one crossing, which consists of twelve class columns.
3. Create one OLAP MDDB in which you import a subset of columns and OLAP summary roles from the group and define eleven crossings.
4. Define a Process Editor Job for the OLAP Group to create the OLAP Table, OLAP MDDB, and a proxy MDDB, which will be used by the HOLAP application to access all the summarized data.

Note: The following explanations describe the metadata and methods used to achieve the desired results. It is assumed that the appropriate Data Warehouse Environment, Data Warehouse, Subject, and Detail Logical Table exist. △

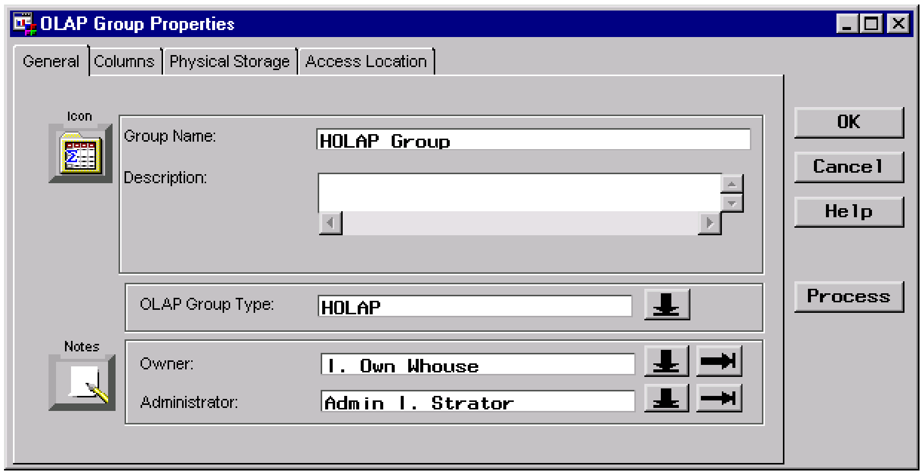
Define OLAP Group Properties

In the SAS/Warehouse Administrator Explorer, position the cursor on the Subject **`Toy Sales`**, click the right mouse button, select **`Add Item`**, and then **`OLAP Group`**. In the Explorer window, a new OLAP Group is added under the Subject as follows:



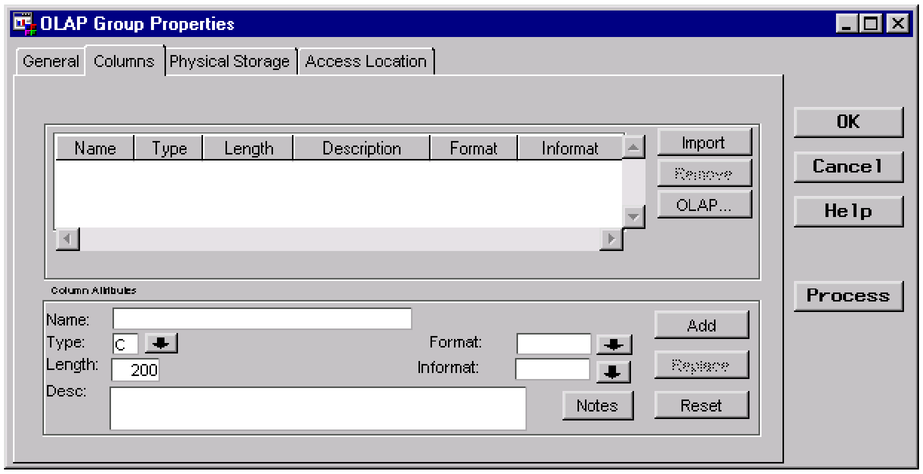
To update the default metadata for the OLAP Group, position the cursor on its icon, click the right mouse button, and select **`Properties`**. The OLAP Group Properties window displays for you to enter the appropriate information.

General Tab specifies the group name **`HOLAP Group`**, description, group type **`HOLAP`**, owner, and administrator.



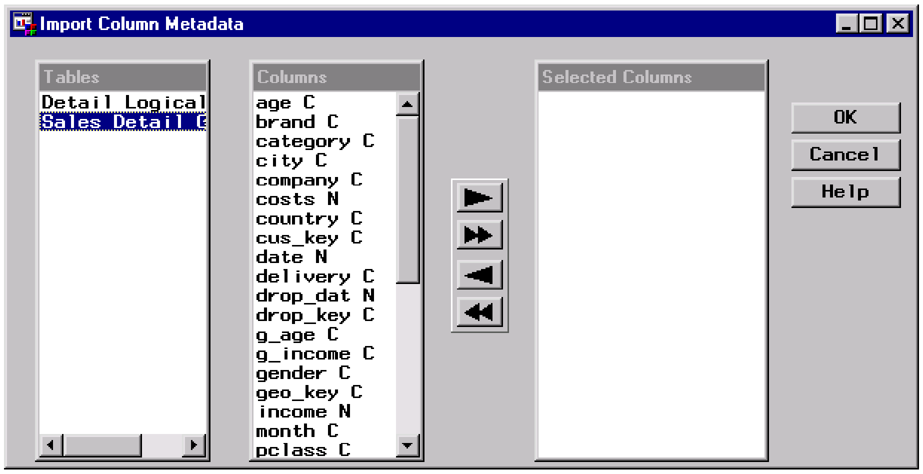
Group Name: HOLAP Group
Description:
OLAP Group Type: HOLAP
Owner: I. Own Whouse
Administrator: Admin I. Strator

Columns Tab specifies the columns to be included in any OLAP summary data store for the OLAP Group, which do not exist yet. When you specify columns at the group level, you are defining the overall OLAP structure so that you can import the columns into specific OLAP Tables and OLAP MDDBs. You generally specify columns at the group level for the HOLAP group type only.

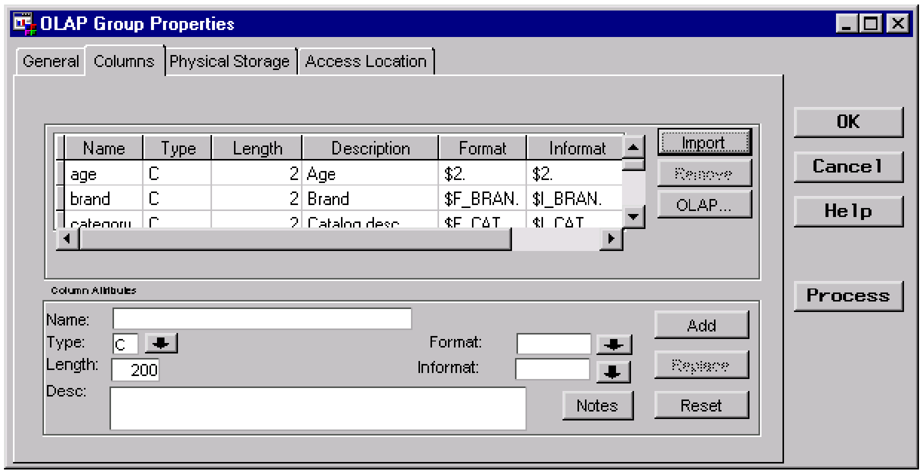


To import columns from an input source, click Import to display a list of input sources, and then select one, for example, **`Detail Logical Tables`**. The Import Column Metadata window displays.

From the Import Column Metadata window, which lists the available input sources, select the Detail Logical Table **`Sales Detail Grp`** to display its available columns.



Select the appropriate columns listed under **`Columns`**, which for this example are all 34 columns, use the double arrows to move them to **`Selected Columns`**, and click OK. You are returned to the Columns tab in the OLAP Group Properties window, which lists the imported columns.

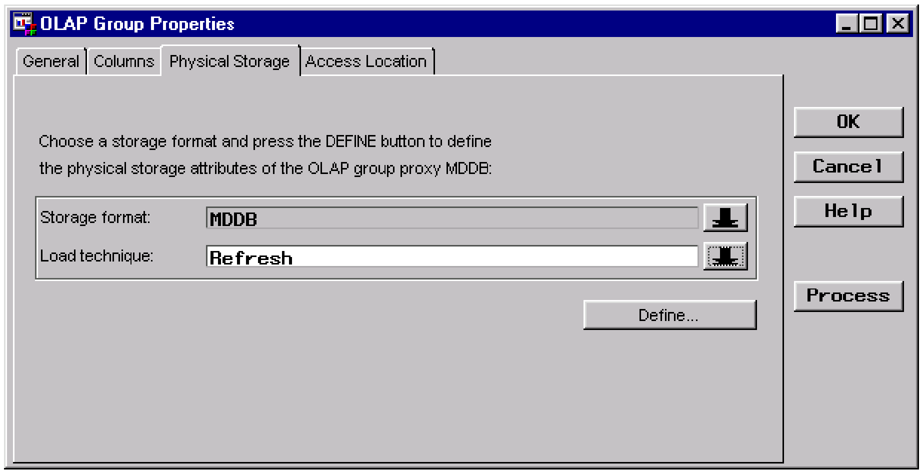


Name	Type	Length	Description	Format	Informat
age	C	2	Age	$2.	$2.
brand	C	2	Brand	$F_BRAN.	$I_BRAN.
category	C	2	Catalog desc	$F_CAT	$I_CAT

Name:
Type: C
Length: 200
Desc:
Format:
Informat:

Physical Storage Tab specifies physical storage attributes. For an OLAP Group, you define physical storage attributes for the HOLAP group type only, which are the attributes for the OLAP Group proxy MDDB.

This example specifies the storage format **`MDDB`** and the load technique **`Refresh`**.

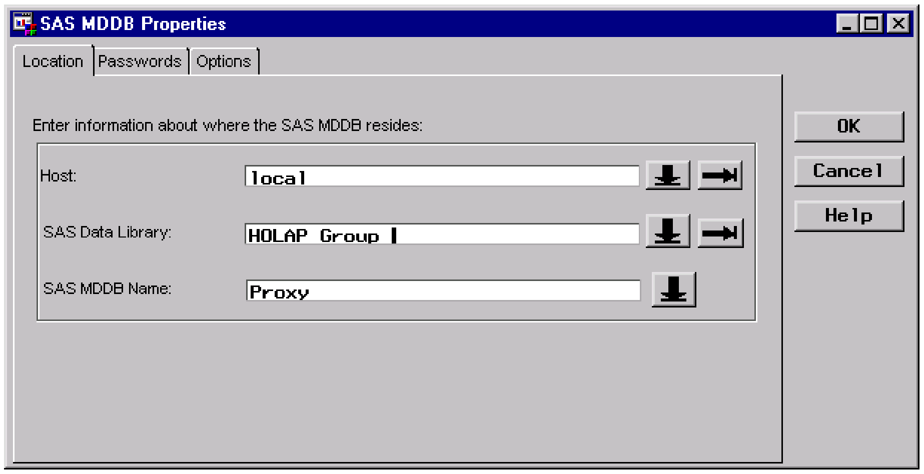


Choose a storage format and press the DEFINE button to define the physical storage attributes of the OLAP group proxy MDDB:

Storage format: MDDB
Load technique: Refresh

Click $\boxed{\text{Define}}$ to open the SAS MDDB Properties window.

Location Tab specifies where the SAS MDDB is stored.



Enter information about where the SAS MDDB resides:

Host: local

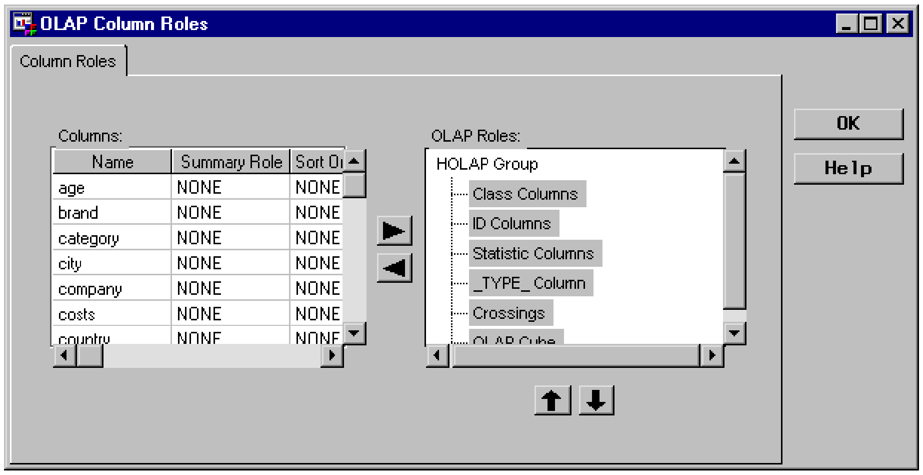
SAS Data Library: HOLAP_Group

SAS MDDB Name: Proxy

Assign OLAP Summary Roles for the Group

Assign Class Columns

From the Columns tab in the OLAP Group Properties window, click OLAP to open the OLAP Column Roles window from which you assign specific OLAP summary roles.



Columns:

Name	Summary Role	Sort Or
age	NONE	NONE
brand	NONE	NONE
category	NONE	NONE
city	NONE	NONE
company	NONE	NONE
costs	NONE	NONE
country	NONE	NONE

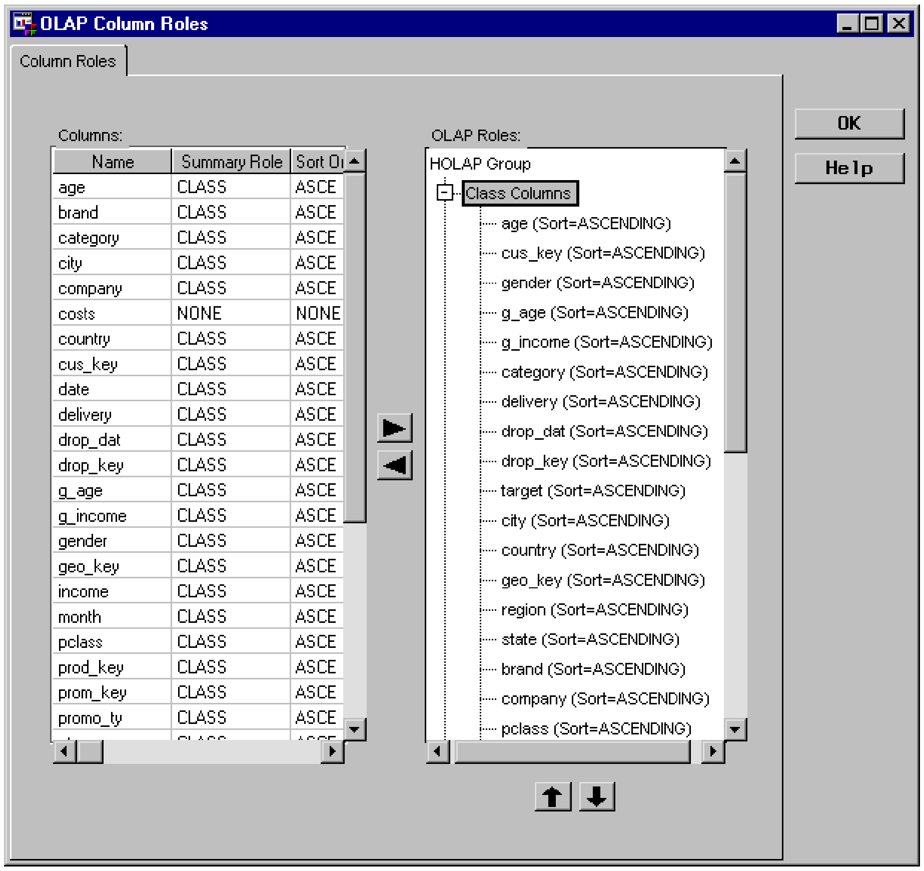
To assign class columns:

1 Select the columns from the **Columns** list.

2 Select the **Class Columns** summary role label in the **OLAP Roles** organization chart.

3 Click the right arrow to add the columns to the summary role.

For this example, class column is assigned to all columns except for three, which will be assigned as statistic columns. The default sort order ASCENDING is used for all class columns.



Columns:

Name	Summary Role	Sort Or
age	CLASS	ASCE
brand	CLASS	ASCE
category	CLASS	ASCE
city	CLASS	ASCE
company	CLASS	ASCE
costs	NONE	NONE
country	CLASS	ASCE
cus_key	CLASS	ASCE
date	CLASS	ASCE
delivery	CLASS	ASCE
drop_dat	CLASS	ASCE
drop_key	CLASS	ASCE
g_age	CLASS	ASCE
g_income	CLASS	ASCE
gender	CLASS	ASCE
geo_key	CLASS	ASCE
income	CLASS	ASCE
month	CLASS	ASCE
pclass	CLASS	ASCE
prod_key	CLASS	ASCE
prom_key	CLASS	ASCE
promo_ty	CLASS	ASCE

OLAP Roles:

```
HOLAP Group
  Class Columns
    age (Sort=ASCENDING)
    cus_key (Sort=ASCENDING)
    gender (Sort=ASCENDING)
    g_age (Sort=ASCENDING)
    g_income (Sort=ASCENDING)
    category (Sort=ASCENDING)
    delivery (Sort=ASCENDING)
    drop_dat (Sort=ASCENDING)
    drop_key (Sort=ASCENDING)
    target (Sort=ASCENDING)
    city (Sort=ASCENDING)
    country (Sort=ASCENDING)
    geo_key (Sort=ASCENDING)
    region (Sort=ASCENDING)
    state (Sort=ASCENDING)
    brand (Sort=ASCENDING)
    company (Sort=ASCENDING)
    pclass (Sort=ASCENDING)
```

Assign Statistic Columns

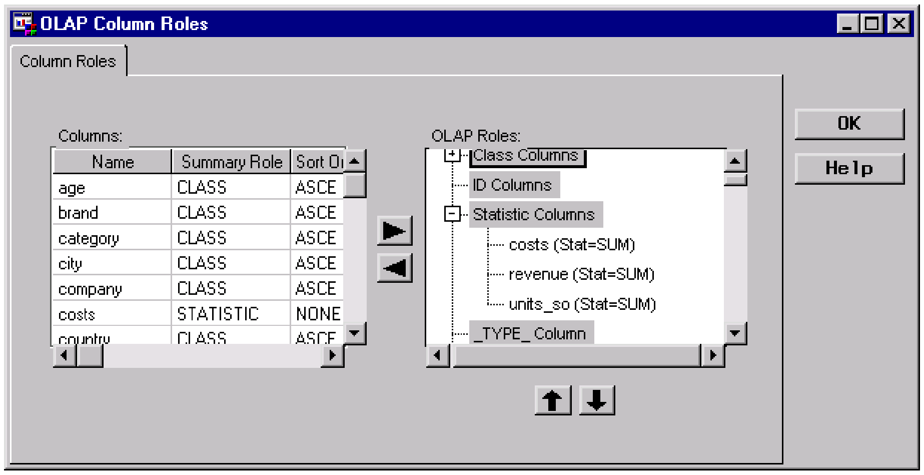
To assign statistic columns:

1 Select the columns from the **`Columns`** list.

2 Select the **`Statistic Columns`** summary role label in the **`OLAP Roles`** organization chart.

3 Click the right arrow to add the columns to the summary role.

For this example, statistic column is assigned to the remaining three columns, which are the numeric columns **`costs`**, **`units_so`**, and **`revenue`**. All three are assigned the default statistic SUM.



Columns:

Name	Summary Role	Sort Or
age	CLASS	ASCE
brand	CLASS	ASCE
category	CLASS	ASCE
city	CLASS	ASCE
company	CLASS	ASCE
costs	STATISTIC	NONE
country	CLASS	ASCE

OLAP Roles:

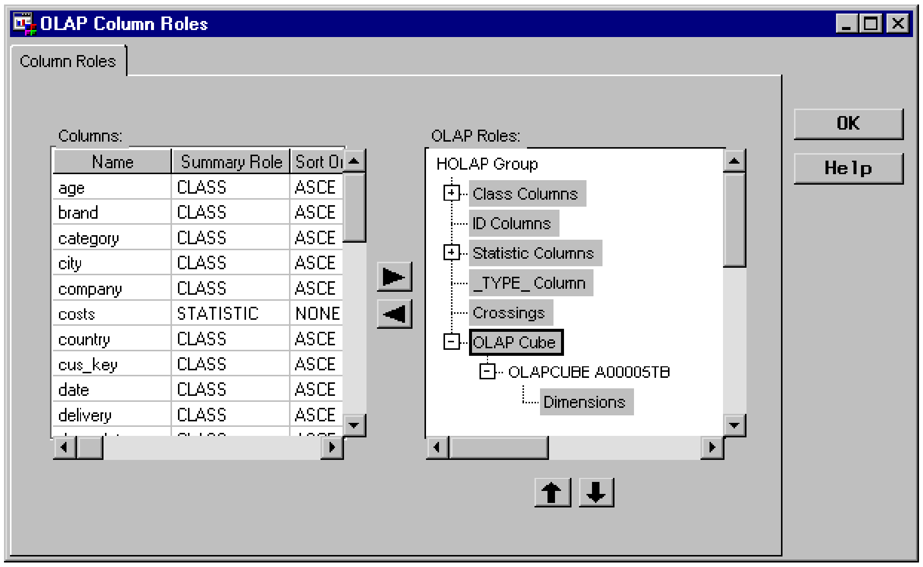
```
  Class Columns
  ID Columns
  Statistic Columns
    costs (Stat=SUM)
    revenue (Stat=SUM)
    units_so (Stat=SUM)
  _TYPE_ Column
```

Define OLAP Structure Definitions for the Group

Add an OLAP Cube

An OLAP Cube is defined for the proxy MDDB, which results in one OLAP Cube for the group describing the relationships of all the class columns. For HOLAP, the OLAP Cube is associated with the OLAP Group and is registered in SAS/EIS software as associated with the proxy MDDB.

In the OLAP Column Roles window, add an OLAP Cube by selecting the **`OLAP Cube`** label in the organization chart. Then click the right mouse button, and select **`New`**.



Columns:

Name	Summary Role	Sort O
age	CLASS	ASCE
brand	CLASS	ASCE
category	CLASS	ASCE
city	CLASS	ASCE
company	CLASS	ASCE
costs	STATISTIC	NONE
country	CLASS	ASCE
cus_key	CLASS	ASCE
date	CLASS	ASCE
delivery	CLASS	ASCE

OLAP Roles:

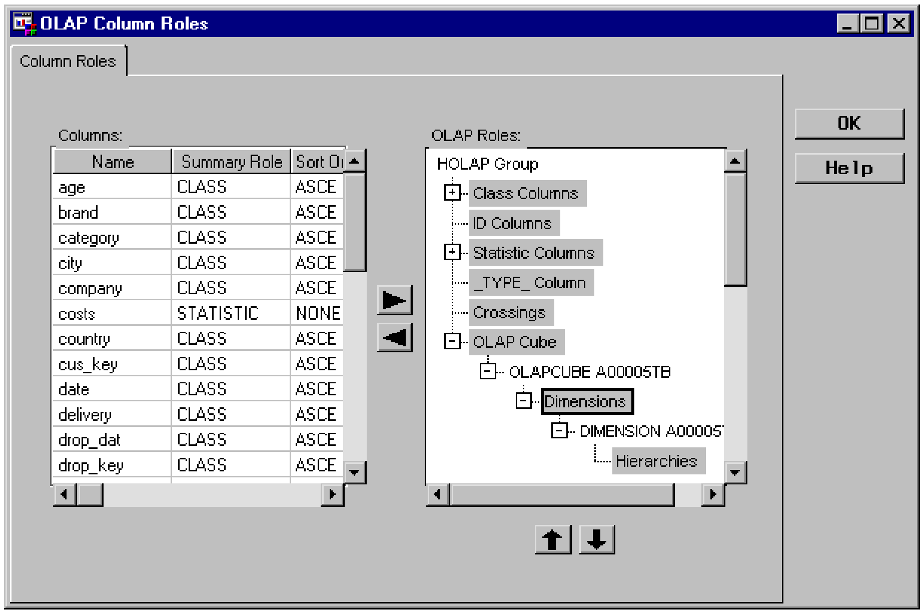
```
HOLAP Group
  Class Columns
  ID Columns
  Statistic Columns
  _TYPE_ Column
  Crossings
  OLAP Cube
    OLAPCUBE A00005TB
      Dimensions
```

Define Dimensions

Dimension objects need to be defined at the group level for each anticipated dimension. As discussed in “Determining Dimensions in the Data” on page 180, this example defines the following dimensions:

- Geography
- Time
- Product
- CustomerGender
- CustomerAge
- CustomerIncome
- Drop
- Promotion

To define a dimension to the OLAP Cube, select the **`Dimensions`** label in the organization chart, click the right mouse button, then select **`New`**.



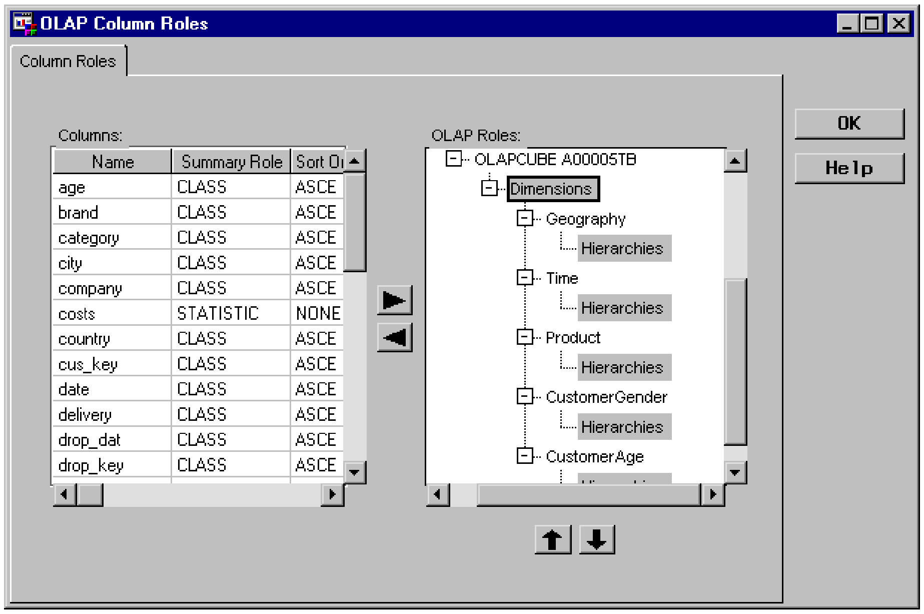
Columns:

Name	Summary Role	Sort Or
age	CLASS	ASCE
brand	CLASS	ASCE
category	CLASS	ASCE
city	CLASS	ASCE
company	CLASS	ASCE
costs	STATISTIC	NONE
country	CLASS	ASCE
cus_key	CLASS	ASCE
date	CLASS	ASCE
delivery	CLASS	ASCE
drop_dat	CLASS	ASCE
drop_key	CLASS	ASCE

OLAP Roles:

```
HOLAP Group
  Class Columns
  ID Columns
  Statistic Columns
  _TYPE_ Column
  Crossings
  OLAP Cube
    OLAPCUBE A00005TB
      Dimensions
        DIMENSION A00005
          Hierarchies
```

To specify a name and a description for a dimension, click the right mouse button on the Dimension object, select **Properties**, and then update the default metadata. The next window displays the defined dimensions for this example:



Columns:

Name	Summary Role	Sort Or
age	CLASS	ASCE
brand	CLASS	ASCE
category	CLASS	ASCE
city	CLASS	ASCE
company	CLASS	ASCE
costs	STATISTIC	NONE
country	CLASS	ASCE
cus_key	CLASS	ASCE
date	CLASS	ASCE
delivery	CLASS	ASCE
drop_dat	CLASS	ASCE
drop_key	CLASS	ASCE

OLAP Roles:

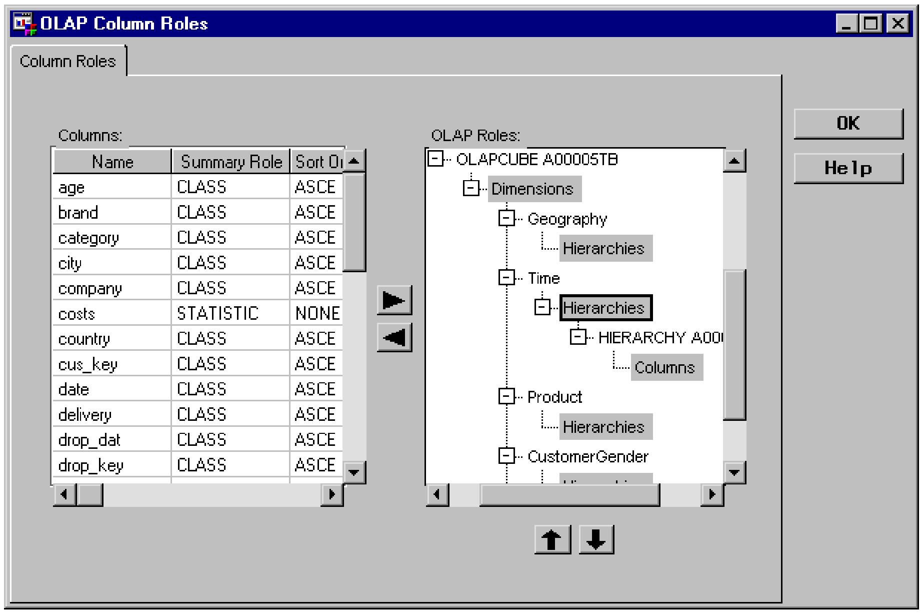
```
OLAPCUBE A00005TB
  Dimensions
    Geography
      Hierarchies
    Time
      Hierarchies
    Product
      Hierarchies
    CustomerGender
      Hierarchies
    CustomerAge
```

Define Hierarchies

Hierarchies are defined for each anticipated hierarchy, which are members of a dimension. For example, the **Time** dimension will include these hierarchies:

- TimeWeek
- TimeQuarter
- TimeSeason

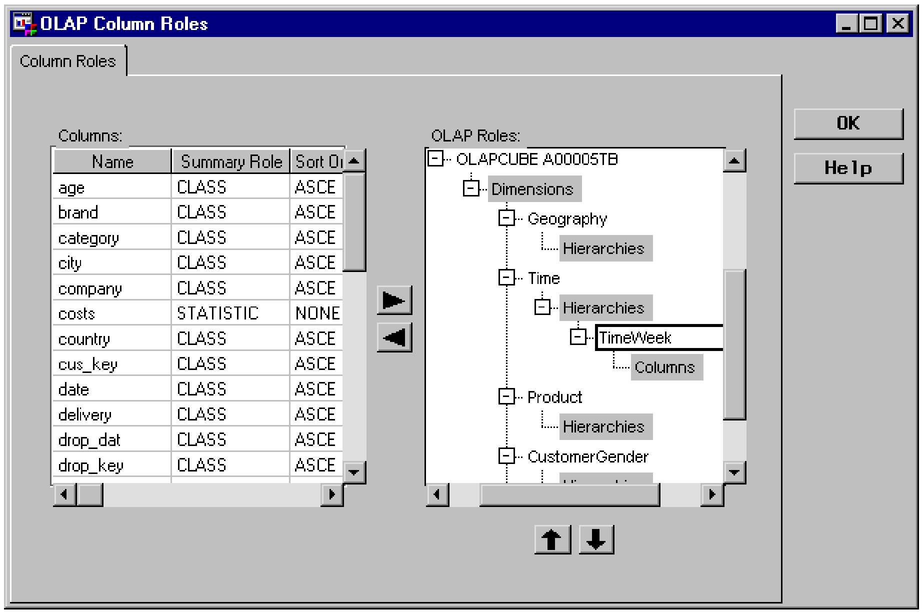
To define a hierarchy for a dimension, select the **Hierarchies** label in the organization chart associated with the appropriate dimension (for example, **Time**) click the right mouse button, and then select **New**.



Name	Summary Role	Sort O
age	CLASS	ASCE
brand	CLASS	ASCE
category	CLASS	ASCE
city	CLASS	ASCE
company	CLASS	ASCE
costs	STATISTIC	NONE
country	CLASS	ASCE
cus_key	CLASS	ASCE
date	CLASS	ASCE
delivery	CLASS	ASCE
drop_dat	CLASS	ASCE
drop_key	CLASS	ASCE

```
OLAPCUBE A00005TB
  Dimensions
    Geography
      Hierarchies
    Time
      Hierarchies
        HIERARCHY A00
          Columns
    Product
      Hierarchies
    CustomerGender
```

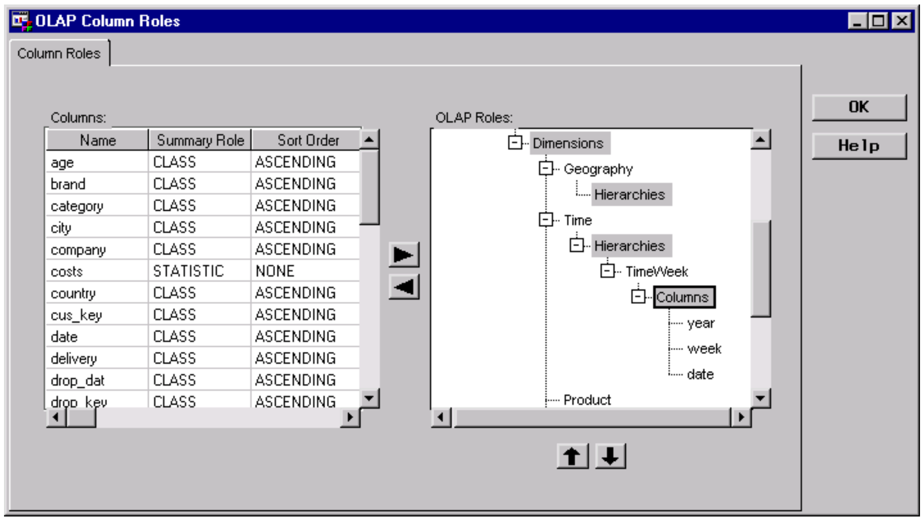
To specify a name (for example, **TimeWeek**) and a description for a hierarchy, click the right mouse button on the Hierarchy object, select **Properties**, and then update the default metadata.



Name	Summary Role	Sort O
age	CLASS	ASCE
brand	CLASS	ASCE
category	CLASS	ASCE
city	CLASS	ASCE
company	CLASS	ASCE
costs	STATISTIC	NONE
country	CLASS	ASCE
cus_key	CLASS	ASCE
date	CLASS	ASCE
delivery	CLASS	ASCE
drop_dat	CLASS	ASCE
drop_key	CLASS	ASCE

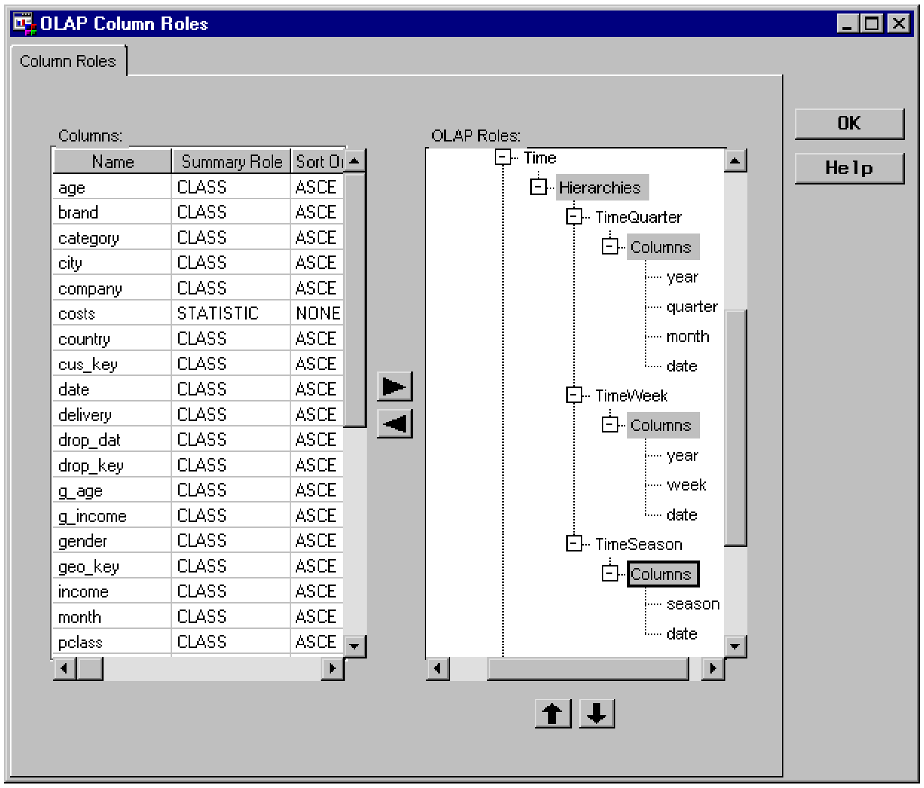
```
OLAPCUBE A00005TB
  Dimensions
    Geography
      Hierarchies
    Time
      Hierarchies
        TimeWeek
          Columns
    Product
      Hierarchies
    CustomerGender
```

Then, to specify the appropriate column(s) for the hierarchy (such as **year**, **week** and **date**) select the class columns from the list under **Columns**, select the **Columns** label associated with the hierarchy object in the organization chart, and click the right arrow to add the columns to the object.



Name	Summary Role	Sort Order
age	CLASS	ASCENDING
brand	CLASS	ASCENDING
category	CLASS	ASCENDING
city	CLASS	ASCENDING
company	CLASS	ASCENDING
costs	STATISTIC	NONE
country	CLASS	ASCENDING
cus_key	CLASS	ASCENDING
date	CLASS	ASCENDING
delivery	CLASS	ASCENDING
drop_dat	CLASS	ASCENDING
drop_key	CLASS	ASCENDING

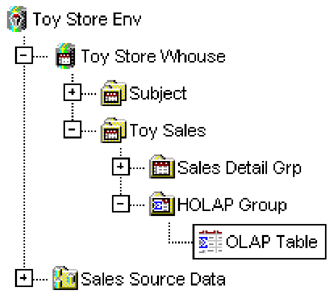
The following window displays the defined hierarchies with appropriate columns for the **`Time`** dimension:



Name	Summary Role	Sort Or
age	CLASS	ASCE
brand	CLASS	ASCE
category	CLASS	ASCE
city	CLASS	ASCE
company	CLASS	ASCE
costs	STATISTIC	NONE
country	CLASS	ASCE
cus_key	CLASS	ASCE
date	CLASS	ASCE
delivery	CLASS	ASCE
drop_dat	CLASS	ASCE
drop_key	CLASS	ASCE
g_age	CLASS	ASCE
g_income	CLASS	ASCE
gender	CLASS	ASCE
geo_key	CLASS	ASCE
income	CLASS	ASCE
month	CLASS	ASCE
pclass	CLASS	ASCE

Define OLAP Table Properties

For the HOLAP application, the OLAP Table will store the largest crossing, which consists of twelve class columns. In the SAS/Warehouse Administrator Explorer, position the cursor on the OLAP Group **`HOLAP Group`**, click the right mouse button, select **`Add Item`**, then **`OLAP Table`**. In the Explorer window, a new OLAP Table is added under the OLAP Group as follows:



To update the metadata for the OLAP Table, position the cursor on its icon, click the right mouse button, and select **`Properties`**. The OLAP Table Properties window displays for you to enter the appropriate information.

General Tab specifies the table name **`Sum 12 OLAP Table`**, description, owner, and administrator.

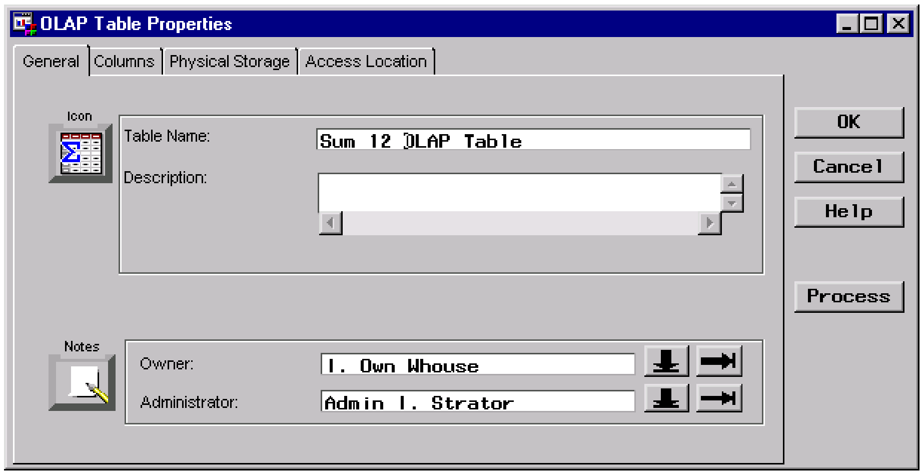


Table Name: Sum 12 OLAP Table

Description:

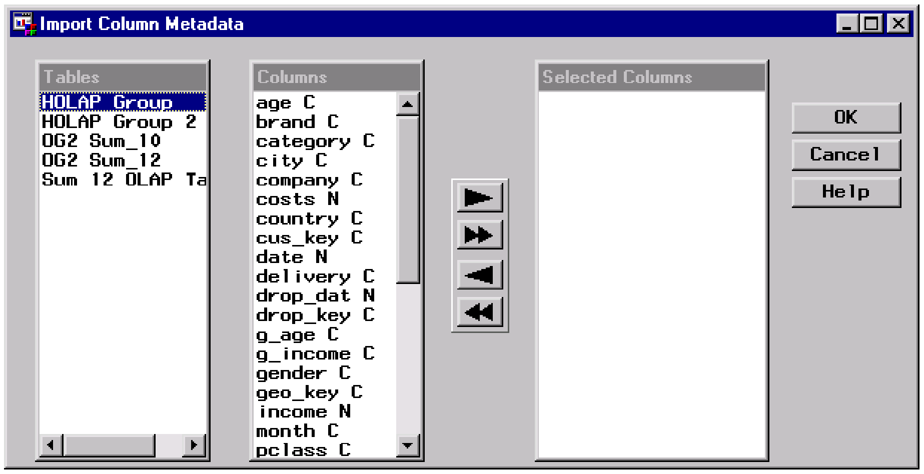
Owner: I. Own Whouse

Administrator: Admin I. Strator

Columns Tab specifies the columns to be included in the OLAP Table. The method is similar to specifying the columns as explained for the OLAP Group; however, for the OLAP Table, the columns are imported from the group.

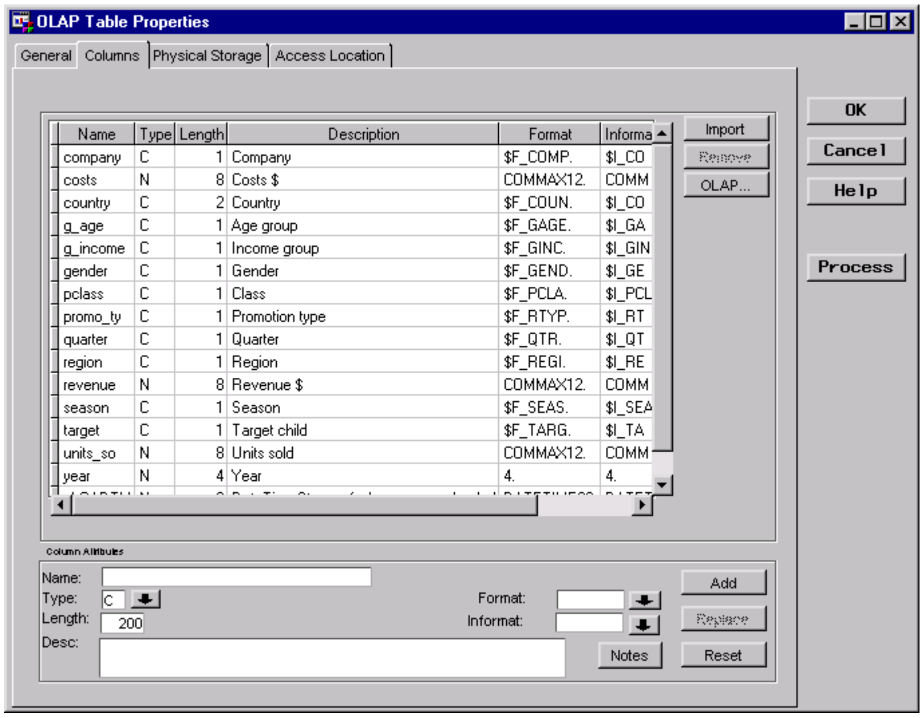
To import the columns from the group, click $\boxed{\text{Import}}$ to display a list of input sources, and then select **`OLAP Data Stores`**. The Import Column Metadata window displays.

From the Import Column Metadata window, which lists the available input sources, select OLAP Group **`HOLAP Group`** to display its available columns.



Tables	Columns
HOLAP Group	age C
HOLAP Group 2	brand C
OG2 Sum_10	category C
OG2 Sum_12	city C
Sum 12 OLAP Ta	company C
	costs N
	country C
	cus_key C
	date N
	delivery C
	drop_dat N
	drop_key C
	g_age C
	g_income C
	gender C
	geo_key C
	income N
	month C
	pclass C

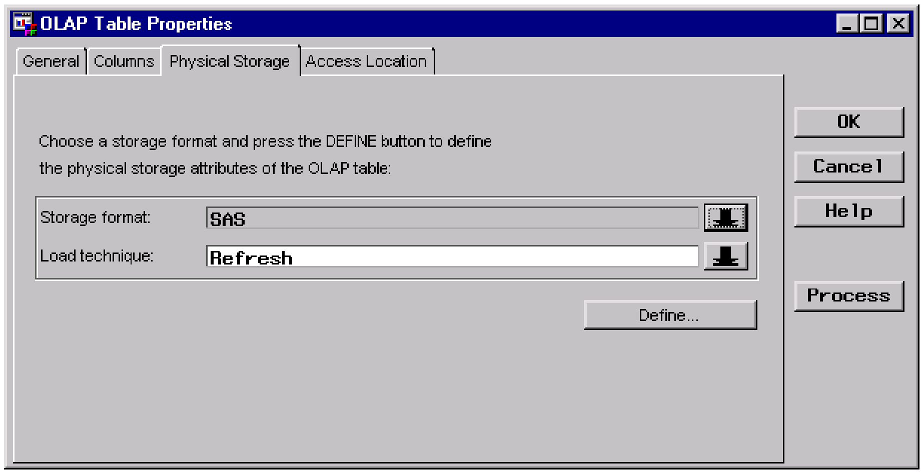
Select the appropriate columns listed under **`Columns`**, which for the OLAP Table are 15 columns from the OLAP Group. Use the double arrows to move them to **`Selected Columns`**, and then click $\boxed{\text{OK}}$. You are returned to the Columns tab in the OLAP Table Properties window, which lists the imported columns as follows:



Name	Type	Length	Description	Format	Informa
company	C	1	Company	$F_COMP.	$I_CO
costs	N	8	Costs $	COMMAX12.	COMM
country	C	2	Country	$F_COUN.	$I_CO
g_age	C	1	Age group	$F_GAGE.	$I_GA
g_income	C	1	Income group	$F_GINC.	$I_GIN
gender	C	1	Gender	$F_GEND.	$I_GE
pclass	C	1	Class	$F_PCLA.	$I_PCL
promo_ty	C	1	Promotion type	$F_RTYP.	$I_RT
quarter	C	1	Quarter	$F_QTR.	$I_QT
region	C	1	Region	$F_REGI.	$I_RE
revenue	N	8	Revenue $	COMMAX12.	COMM
season	C	1	Season	$F_SEAS.	$I_SEA
target	C	1	Target child	$F_TARG.	$I_TA
units_so	N	8	Units sold	COMMAX12.	COMM
year	N	4	Year	4.	4.

Name:			
Type:	C	Format:	
Length:	200	Informat:	
Desc:			

Physical Storage Tab
: specifies physical storage attributes for the OLAP Table. This example specifies the storage format **`SAS`** and the load technique **`Refresh`**.



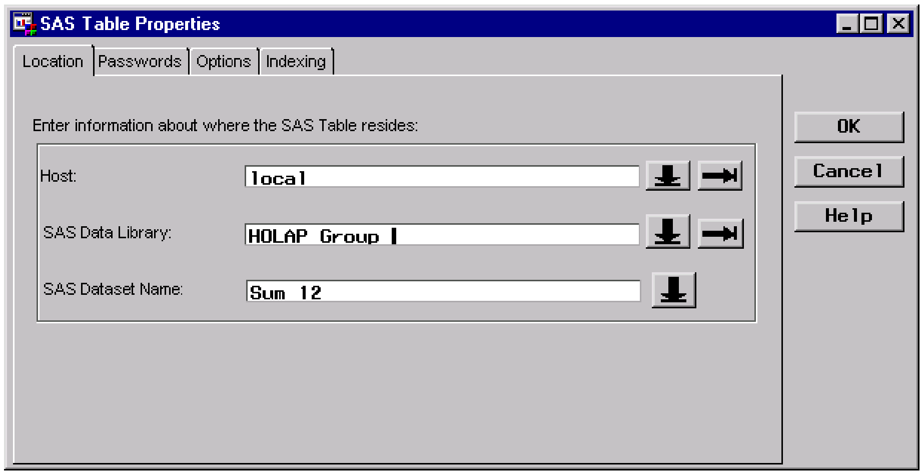
Choose a storage format and press the DEFINE button to define the physical storage attributes of the OLAP table:

Storage format: SAS

Load technique: Refresh

Click Define to open the SAS Table Properties window.

Location Tab specifies where the SAS Table is stored.



Enter information about where the SAS Table resides:

Host: local

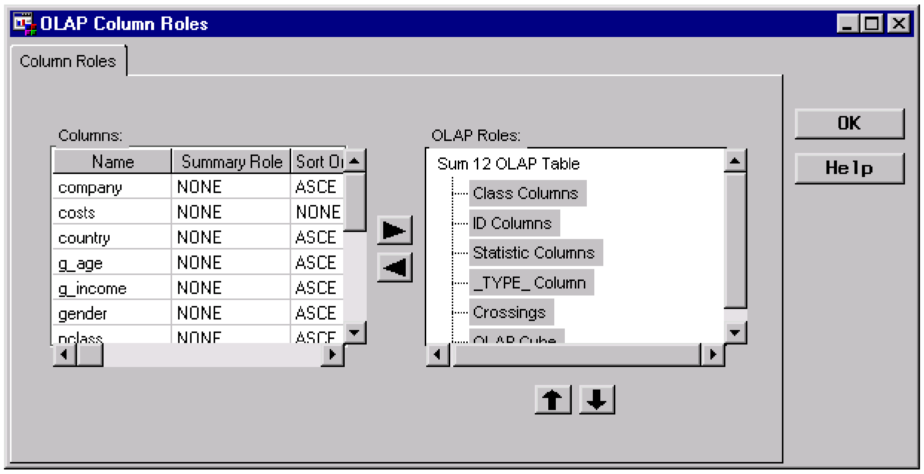
SAS Data Library: HOLAP Group

SAS Dataset Name: Sum 12

Import OLAP Summary Roles for the Table

Import Class Columns

This example imports the summary roles already assigned for the OLAP Group. From the Columns tab in the OLAP Table Properties window, click OLAP to open the OLAP Column Roles window.



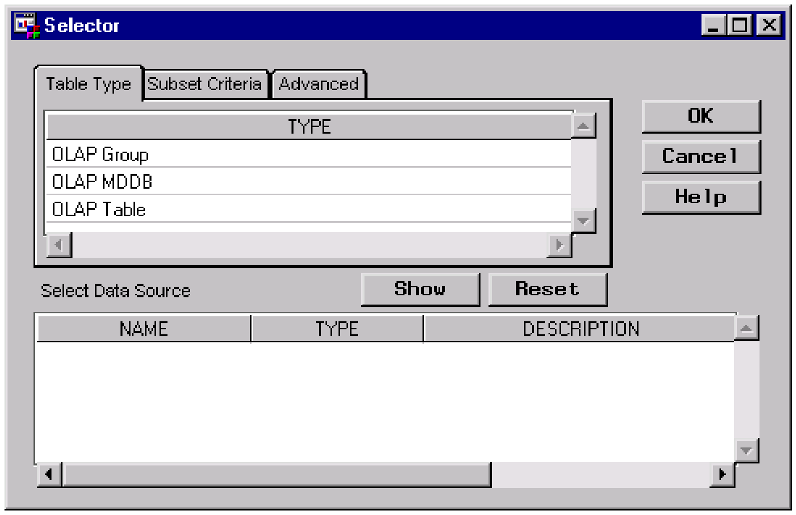
Columns:

Name	Summary Role	Sort O
company	NONE	ASCE
costs	NONE	NONE
country	NONE	ASCE
g_age	NONE	ASCE
g_income	NONE	ASCE
gender	NONE	ASCE
nclass	NONE	ASCE

OLAP Roles:

- Sum 12 OLAP Table
  - Class Columns
  - ID Columns
  - Statistic Columns
  - _TYPE_ Column
  - Crossings
  - OLAP Cube

To import the class columns, in the organization chart, click the right mouse button on the **`Class Columns`** label and select **`Import`**. The software opens the Selector window.

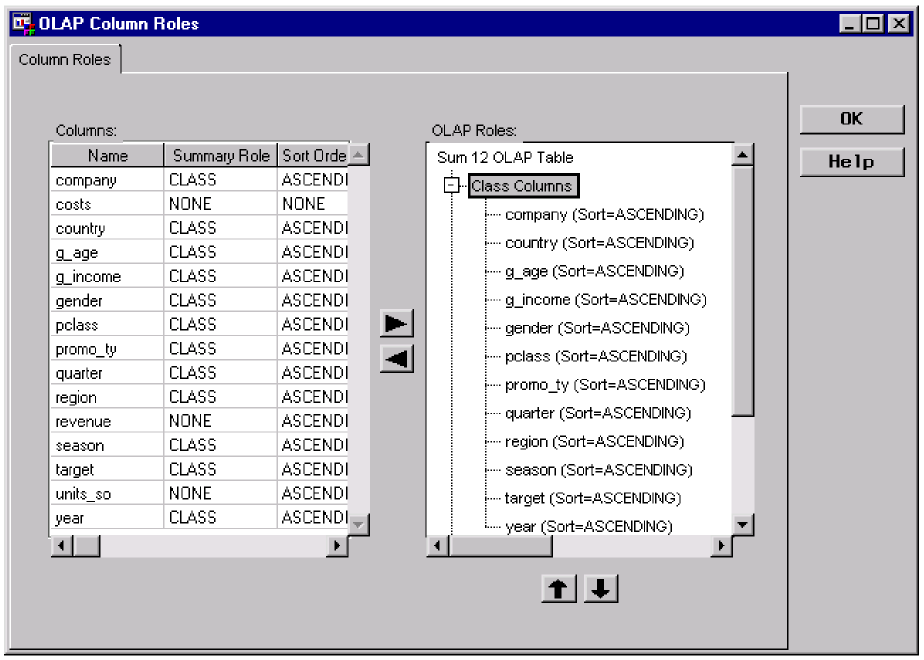


TYPE
OLAP Group
OLAP MDDB
OLAP Table

NAME	TYPE	DESCRIPTION

Select **`OLAP Group`**, click [Show], select **`HOLAP Group`**, and then click [OK]. The software imports the class columns, and you are returned to the OLAP Column Roles window.

Note: For each column contained in the group that is not specified for the table, a message will display requiring you to click [OK]. △



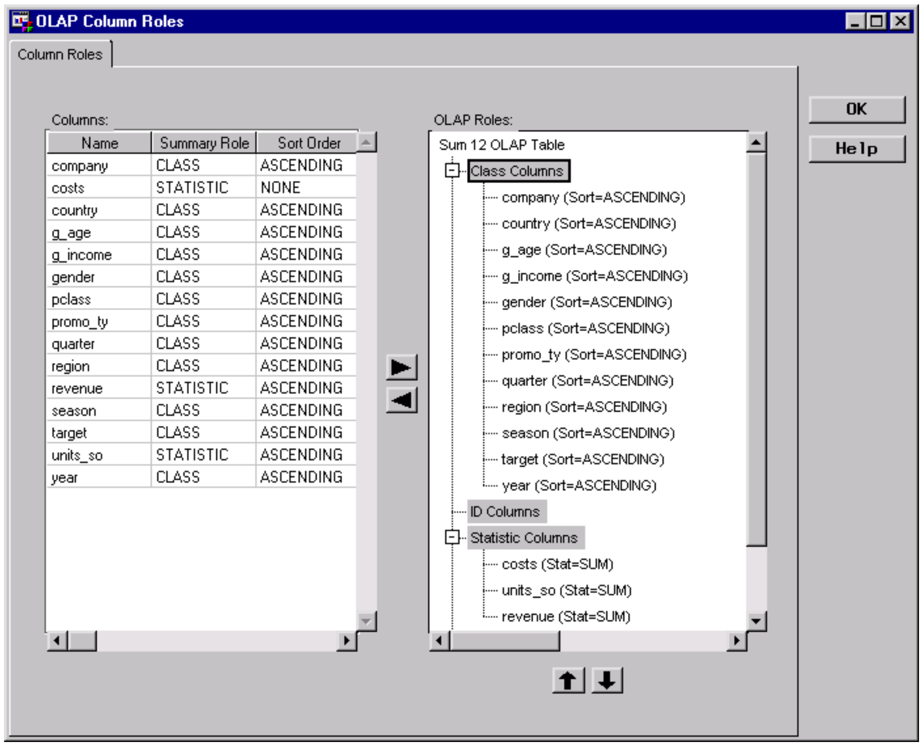
Name	Summary Role	Sort Orde
company	CLASS	ASCENDI
costs	NONE	NONE
country	CLASS	ASCENDI
g_age	CLASS	ASCENDI
g_income	CLASS	ASCENDI
gender	CLASS	ASCENDI
pclass	CLASS	ASCENDI
promo_ty	CLASS	ASCENDI
quarter	CLASS	ASCENDI
region	CLASS	ASCENDI
revenue	NONE	ASCENDI
season	CLASS	ASCENDI
target	CLASS	ASCENDI
units_so	NONE	ASCENDI
year	CLASS	ASCENDI

```
Sum 12 OLAP Table
 Class Columns
    company (Sort=ASCENDING)
    country (Sort=ASCENDING)
    g_age (Sort=ASCENDING)
    g_income (Sort=ASCENDING)
    gender (Sort=ASCENDING)
    pclass (Sort=ASCENDING)
    promo_ty (Sort=ASCENDING)
    quarter (Sort=ASCENDING)
    region (Sort=ASCENDING)
    season (Sort=ASCENDING)
    target (Sort=ASCENDING)
    year (Sort=ASCENDING)
```

Import Statistic Columns

To import the statistic columns, follow the same steps as explained for importing class columns. That is, in the organization chart, click the right mouse button on the **Statistic Columns** label and select **Import**. The software opens the Selector window.

From the Selector window, select **OLAP Group**, click Show, select **HOLAP Group**, then click OK. The software imports the statistic columns, and returns you to the OLAP Column Roles window as follows:



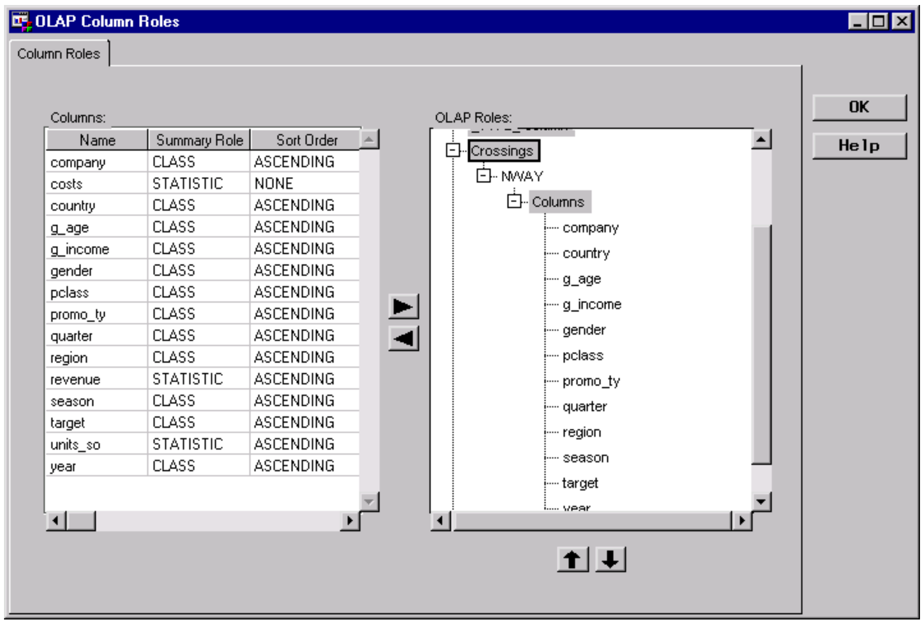
Name	Summary Role	Sort Order
company	CLASS	ASCENDING
costs	STATISTIC	NONE
country	CLASS	ASCENDING
g_age	CLASS	ASCENDING
g_income	CLASS	ASCENDING
gender	CLASS	ASCENDING
pclass	CLASS	ASCENDING
promo_ty	CLASS	ASCENDING
quarter	CLASS	ASCENDING
region	CLASS	ASCENDING
revenue	STATISTIC	ASCENDING
season	CLASS	ASCENDING
target	CLASS	ASCENDING
units_so	STATISTIC	ASCENDING
year	CLASS	ASCENDING

```
Sum 12 OLAP Table
 Class Columns
    company (Sort=ASCENDING)
    country (Sort=ASCENDING)
    g_age (Sort=ASCENDING)
    g_income (Sort=ASCENDING)
    gender (Sort=ASCENDING)
    pclass (Sort=ASCENDING)
    promo_ty (Sort=ASCENDING)
    quarter (Sort=ASCENDING)
    region (Sort=ASCENDING)
    season (Sort=ASCENDING)
    target (Sort=ASCENDING)
    year (Sort=ASCENDING)
 ID Columns
 Statistic Columns
    costs (Stat=SUM)
    units_so (Stat=SUM)
    revenue (Stat=SUM)
```

Define Crossing for the Table

For the HOLAP application, the OLAP Table will store the largest crossing, which consists of 12 class columns. (For an explanation about how these 12 class columns were determined, see "Determining Crossing(s)" on page 181.) The crossing is defined for the OLAP Table as an NWAY crossing, which is a crossing that consists of all the assigned class columns.

To define the NWAY crossing from the OLAP Column Roles window, in the organization chart on the **`Crossings`** label, click the right mouse button, and select **`Create NWAY Crossing`**. The resulting NWAY crossing follows:



Columns:

Name	Summary Role	Sort Order
company	CLASS	ASCENDING
costs	STATISTIC	NONE
country	CLASS	ASCENDING
g_age	CLASS	ASCENDING
g_income	CLASS	ASCENDING
gender	CLASS	ASCENDING
pclass	CLASS	ASCENDING
promo_ty	CLASS	ASCENDING
quarter	CLASS	ASCENDING
region	CLASS	ASCENDING
revenue	STATISTIC	ASCENDING
season	CLASS	ASCENDING
target	CLASS	ASCENDING
units_so	STATISTIC	ASCENDING
year	CLASS	ASCENDING

OLAP Roles:

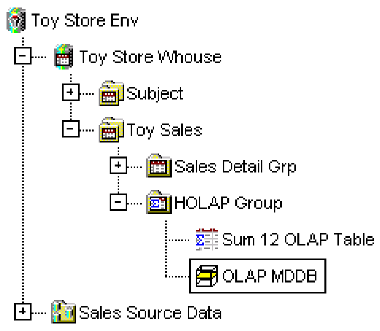
```
Crossings
  NWAY
    Columns
      company
      country
      g_age
      g_income
      gender
      pclass
      promo_ty
      quarter
      region
      season
      target
      year
```

Note: For a HOLAP application, you do not need to create an NWAY crossing of all the class columns defined for the OLAP Group, which consists of 31 columns. Because the detail data is available from a permanent data store, the HOLAP application can access all the detail data without it being referenced in a crossing. △

Note: If an OLAP Table has multiple crossings, the _TYPE_ column will store numeric values that identify which class columns are included in each crossing. If you define more than one crossing for an OLAP Table and do not define a column for this role, SAS/Warehouse Administrator will display a dialog asking whether a column named _TYPE_ can be added. If you say YES, the column will appear in the Columns table and will also be assigned the summary role of _TYPE_. You can rename the column. For an explanation of the values for _TYPE_, see the MEANS procedure in the *SAS Procedures Guide*. △

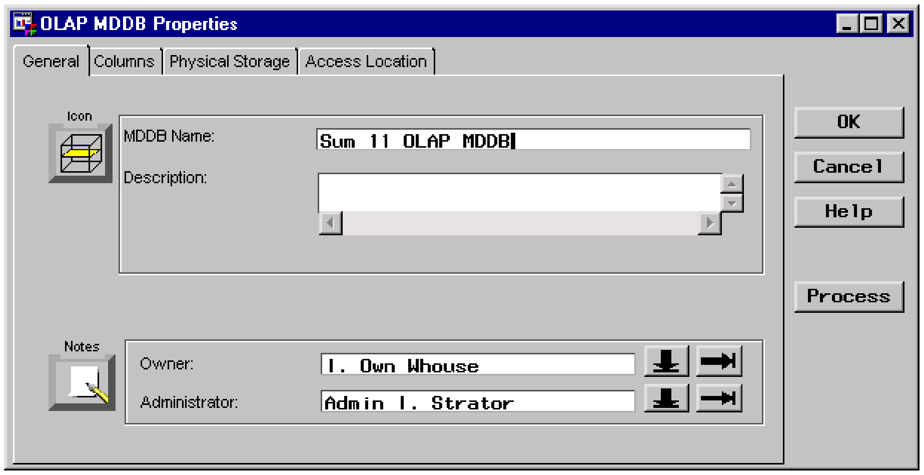
Define OLAP MDDB Properties

For the HOLAP application, the OLAP MDDB will store 11 crossings. In the SAS/Warehouse Administrator Explorer, position the cursor on the OLAP Group **`HOLAP Group`**, click the right mouse button, select **`Add Item`**, and then **`OLAP MDDB`**. In the Explorer window, a new OLAP MDDB is added under the OLAP Group as follows:



To update the metadata for the OLAP MDDB, position the cursor on its icon, click the right mouse button, and select **`Properties`**. The OLAP MDDB Properties window displays for you to enter the appropriate information.

General Tab — specifies the MDDB name **`Sum 11 OLAP MDDB`**, description, owner, and administrator.

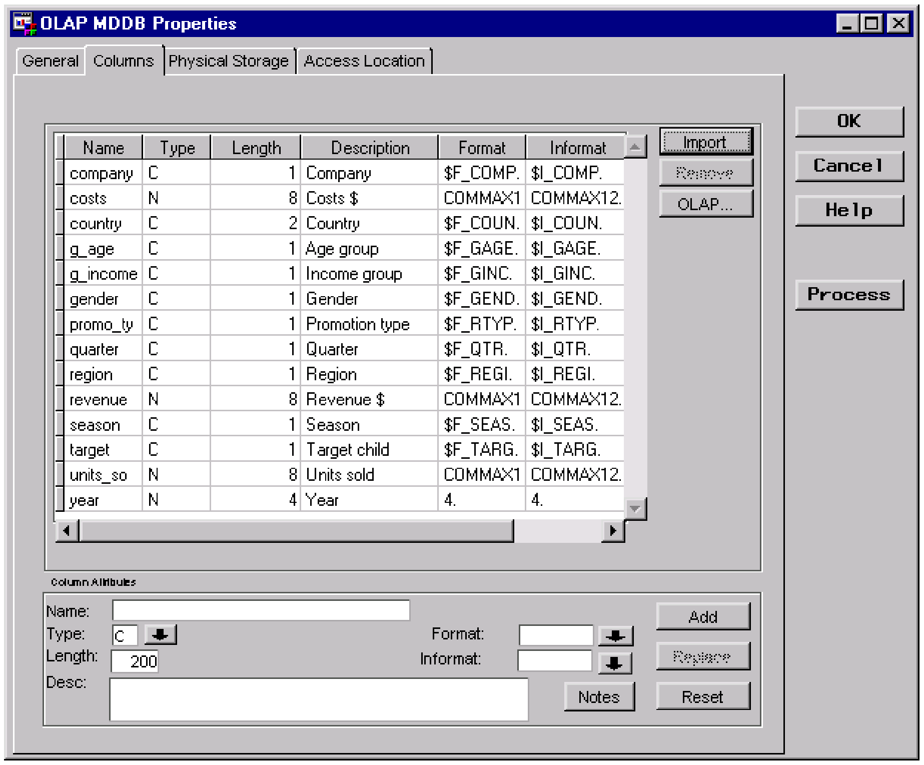


MDDB Name: Sum 11 OLAP MDDB
Description:
Owner: I. Own Whouse
Administrator: Admin I. Strator

Columns Tab — specifies the columns to be included in the OLAP MDDB. To import the columns from the group is identical to the method explained for the OLAP Table. See "Define OLAP Table Properties" on page 195.

Select the appropriate column definitions, which for this example imports 14 columns for the OLAP MDDB from the OLAP Group. Note that **`pclass`** is the column included in the OLAP Table but not in the OLAP MDDB.

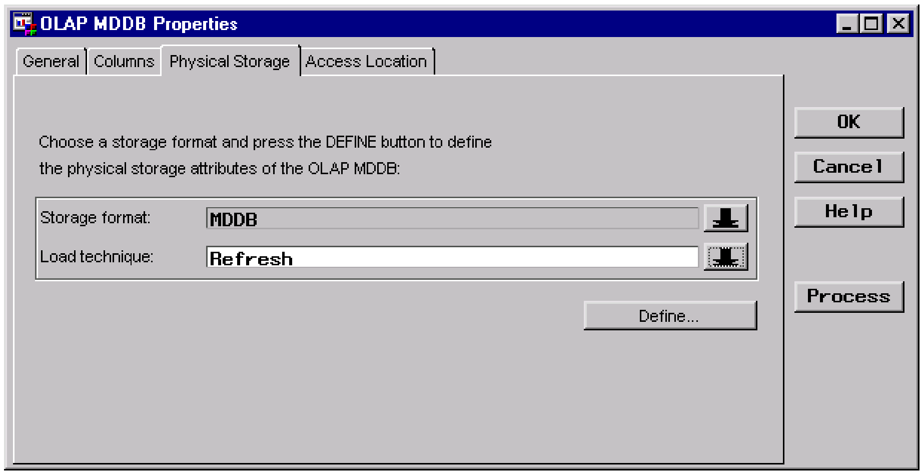
The following window shows the imported columns for the OLAP MDDB:



Name	Type	Length	Description	Format	Informat
company	C	1	Company	$F_COMP.	$I_COMP.
costs	N	8	Costs $	COMMAX1	COMMAX12.
country	C	2	Country	$F_COUN.	$I_COUN.
g_age	C	1	Age group	$F_GAGE.	$I_GAGE.
g_income	C	1	Income group	$F_GINC.	$I_GINC.
gender	C	1	Gender	$F_GEND.	$I_GEND.
promo_ty	C	1	Promotion type	$F_RTYP.	$I_RTYP.
quarter	C	1	Quarter	$F_QTR.	$I_QTR.
region	C	1	Region	$F_REGI.	$I_REGI.
revenue	N	8	Revenue $	COMMAX1	COMMAX12.
season	C	1	Season	$F_SEAS.	$I_SEAS.
target	C	1	Target child	$F_TARG.	$I_TARG.
units_so	N	8	Units sold	COMMAX1	COMMAX12.
year	N	4	Year	4.	4.

Name:
Type: C
Length: 200
Desc:
Format:
Informat:

Physical Storage Tab
specifies the physical storage attributes for the OLAP MDDB. This example specifies the storage format **MDDB** and the load technique **Refresh**.

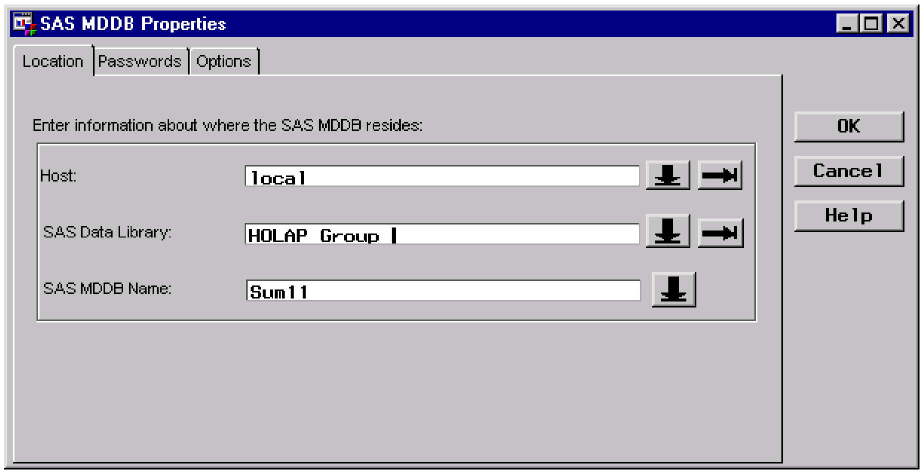


Choose a storage format and press the DEFINE button to define the physical storage attributes of the OLAP MDDB:

Storage format: MDDB
Load technique: Refresh

Click Define to open the SAS MDDB Properties window.

Location Tab specifies where the SAS MDDB is stored.



Enter information about where the SAS MDDB resides:

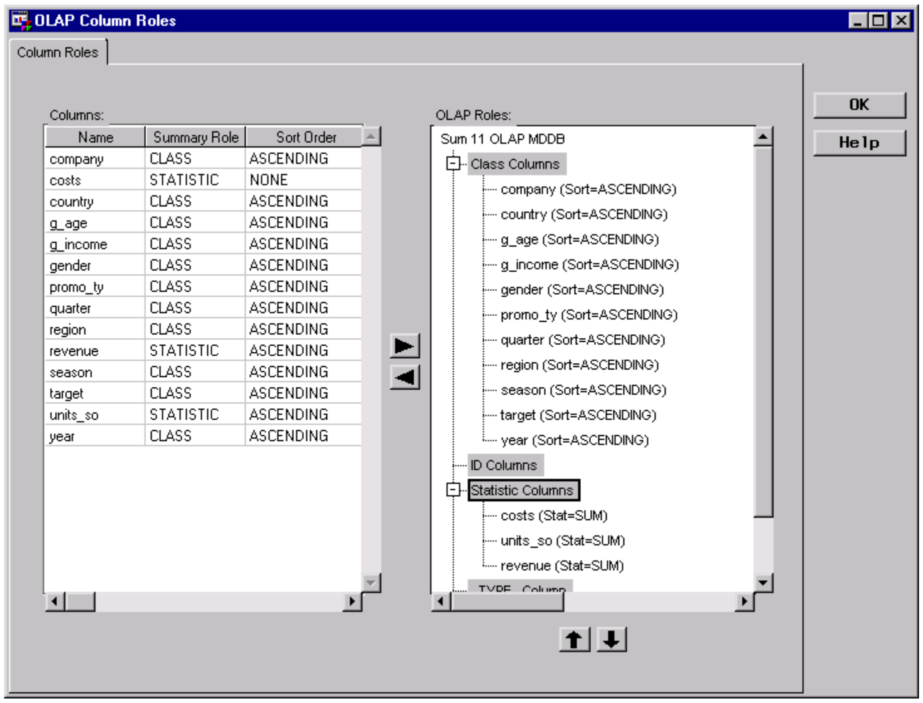
Host: local

SAS Data Library: HOLAP Group I

SAS MDDB Name: Sum11

Import OLAP Summary Roles for the MDDB

To import the class columns and statistic columns, use the same steps as explained for the OLAP Table. See "Import OLAP Summary Roles for the Table" on page 198. The following window displays the imported class columns and statistic columns for the OLAP MDDB:



Columns:

Name	Summary Role	Sort Order
company	CLASS	ASCENDING
costs	STATISTIC	NONE
country	CLASS	ASCENDING
g_age	CLASS	ASCENDING
g_income	CLASS	ASCENDING
gender	CLASS	ASCENDING
promo_ty	CLASS	ASCENDING
quarter	CLASS	ASCENDING
region	CLASS	ASCENDING
revenue	STATISTIC	ASCENDING
season	CLASS	ASCENDING
target	CLASS	ASCENDING
units_so	STATISTIC	ASCENDING
year	CLASS	ASCENDING

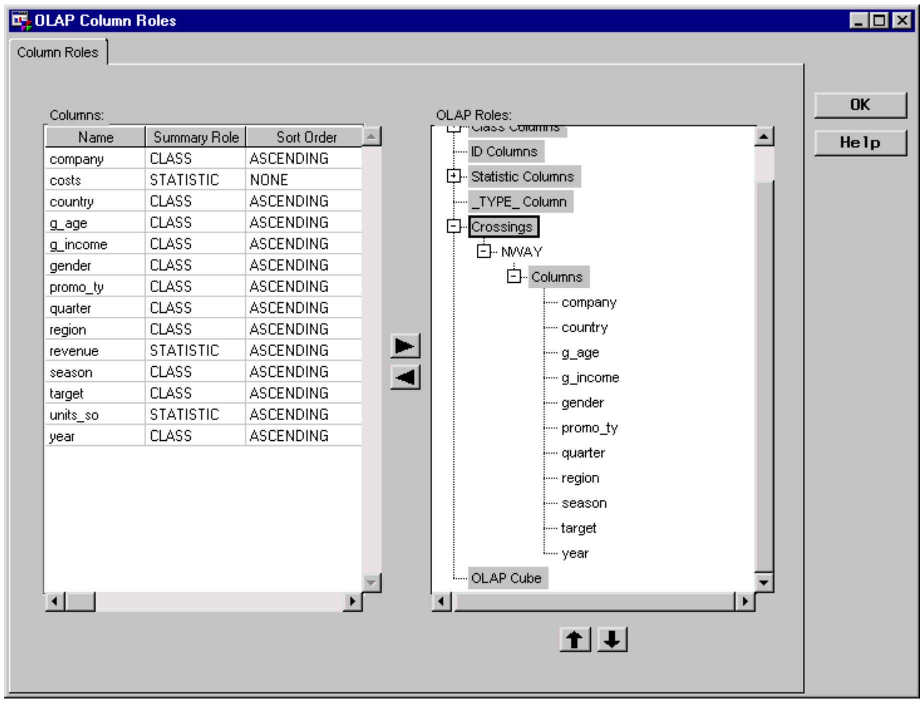
OLAP Roles:

```
Sum 11 OLAP MDDB
  Class Columns
    company (Sort=ASCENDING)
    country (Sort=ASCENDING)
    g_age (Sort=ASCENDING)
    g_income (Sort=ASCENDING)
    gender (Sort=ASCENDING)
    promo_ty (Sort=ASCENDING)
    quarter (Sort=ASCENDING)
    region (Sort=ASCENDING)
    season (Sort=ASCENDING)
    target (Sort=ASCENDING)
    year (Sort=ASCENDING)
  ID Columns
  Statistic Columns
    costs (Stat=SUM)
    units_so (Stat=SUM)
    revenue (Stat=SUM)
  TYPE_Column
```

Define Crossings for the MDDB

For the HOLAP application, the OLAP MDDB will store 11 crossings. The crossings are defined for the OLAP MDDB first as an NWAY crossing, then by creating stairstep crossings from the NWAY crossing.

To define the NWAY crossing, in the organization chart on the **`Crossings`** label, click the right mouse button, and select **`Create NWAY Crossing`**. The resulting NWAY crossing follows:



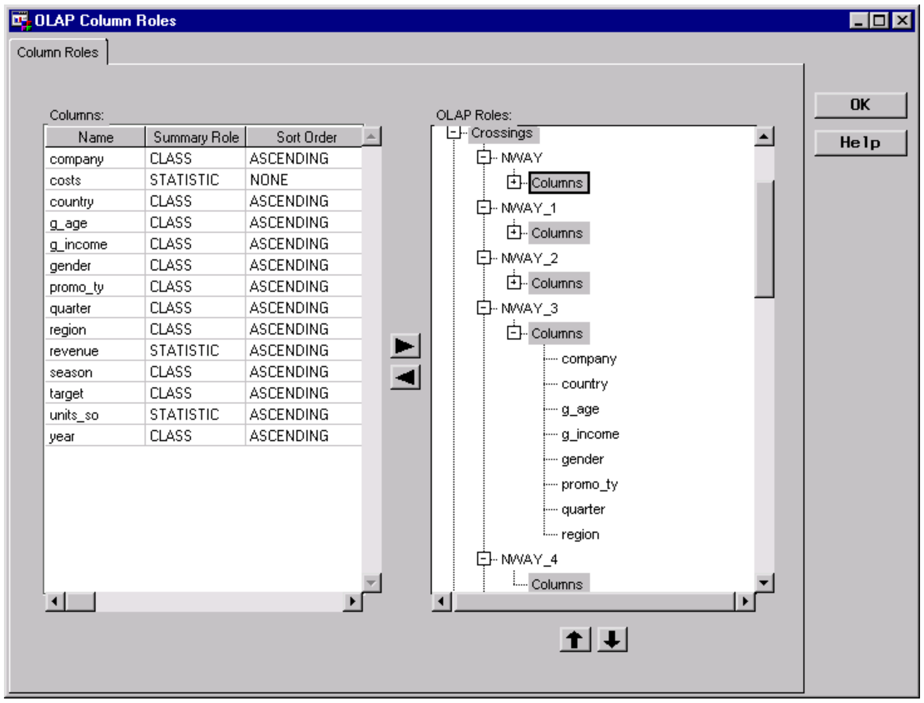
Columns:

Name	Summary Role	Sort Order
company	CLASS	ASCENDING
costs	STATISTIC	NONE
country	CLASS	ASCENDING
g_age	CLASS	ASCENDING
g_income	CLASS	ASCENDING
gender	CLASS	ASCENDING
promo_ty	CLASS	ASCENDING
quarter	CLASS	ASCENDING
region	CLASS	ASCENDING
revenue	STATISTIC	ASCENDING
season	CLASS	ASCENDING
target	CLASS	ASCENDING
units_so	STATISTIC	ASCENDING
year	CLASS	ASCENDING

OLAP Roles:

```
Class Columns
ID Columns
[+] Statistic Columns
_TYPE_ Column
[-] Crossings
    [-] NWAY
        [-] Columns
            company
            country
            g_age
            g_income
            gender
            promo_ty
            quarter
            region
            season
            target
            year
OLAP Cube
```

To create the remaining 10 crossings, in the organization chart on the NWAY crossing, click the right mouse button, and select **`Create Stairstep Crossings`**. Ten crossings are added, with each one having one less column than the crossing before it as follows:



Columns:

Name	Summary Role	Sort Order
company	CLASS	ASCENDING
costs	STATISTIC	NONE
country	CLASS	ASCENDING
g_age	CLASS	ASCENDING
g_income	CLASS	ASCENDING
gender	CLASS	ASCENDING
promo_ty	CLASS	ASCENDING
quarter	CLASS	ASCENDING
region	CLASS	ASCENDING
revenue	STATISTIC	ASCENDING
season	CLASS	ASCENDING
target	CLASS	ASCENDING
units_so	STATISTIC	ASCENDING
year	CLASS	ASCENDING

OLAP Roles:

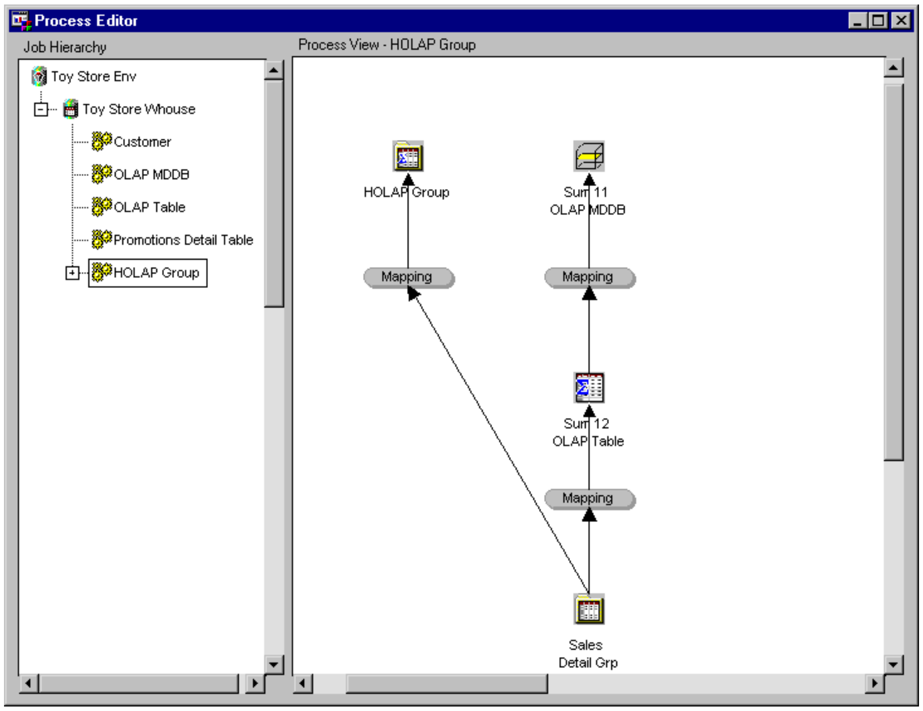
```
[-] Crossings
    [-] NWAY
        [+] Columns
    [-] NWAY_1
        [+] Columns
    [-] NWAY_2
        [+] Columns
    [-] NWAY_3
        [-] Columns
            company
            country
            g_age
            g_income
            gender
            promo_ty
            quarter
            region
    [-] NWAY_4
        Columns
```

Define Process Editor Job

For the Process Editor Job, the OLAP Group and all its child summary data stores are specified as output targets in a single Job. The Detail Logical Table **`Sales Detail Grp`** is specified as the input source for the two summary data stores as well as the OLAP Group.

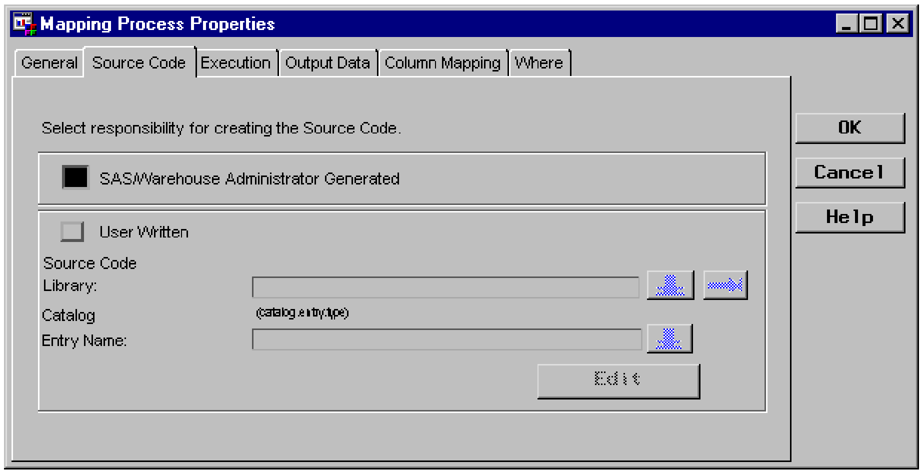
When an OLAP Group of type HOLAP is specified as an output target, SAS/Warehouse Administrator generates a proxy MDDB. The proxy MDDB is an empty physical file that represents the structure of the data in an OLAP Group and can be used by SAS/EIS software to provide more efficient access to multiple OLAP Tables and OLAP MDDBs. Note that the proxy MDDB automatically creates an NWAY crossing to be used by the HOLAP application.

The following Process Editor window displays the Process Flow for the Job:



The processes defined in the Job are summarized as follows:

Mapping Process	□ The source code to map columns is generated by SAS/Warehouse Administrator, rather than user-written, as shown in the Source Code tab for the Mapping Process Properties window:



Select responsibility for creating the Source Code.

■ SAS/Warehouse Administrator Generated

☐ User Written

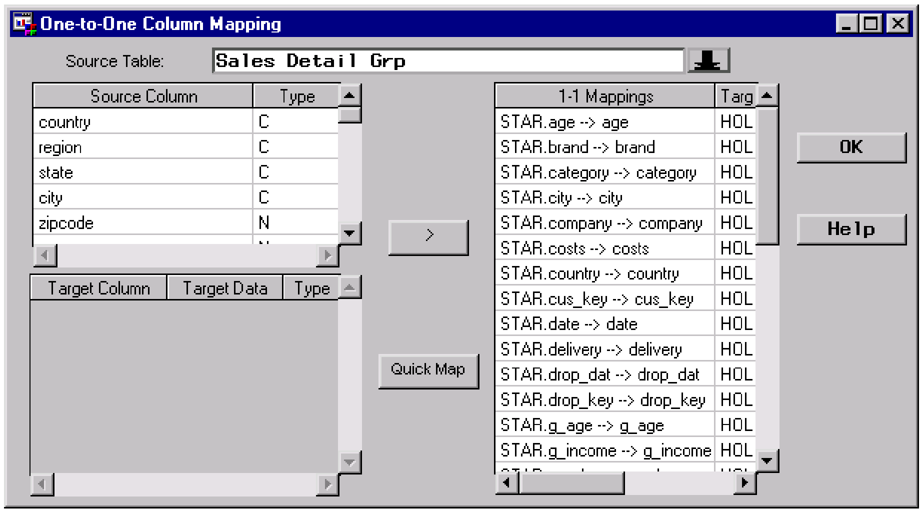
Source Code

Library:

Catalog (catalog.entry.type)

Entry Name:

□ Column mapping is defined as one-to-one mapping. To produce one-to-one mapping, first click $\boxed{\text{1 to 1 Mappings}}$ on the Column Mapping tab in the Mapping Process Properties window, which opens the One-to-One Column Mapping window. Then, click $\boxed{\text{Quick Map}}$ to create the one-to-one mapping as follows:



Source Table: Sales Detail Grp

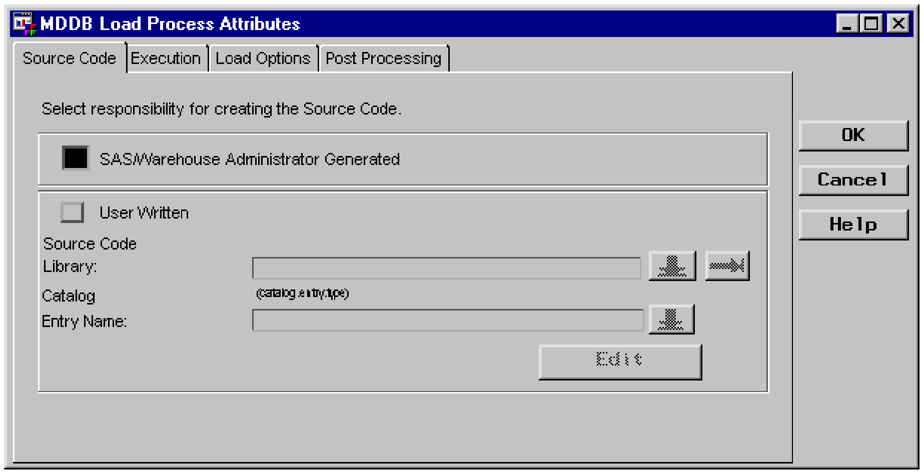
Source Column	Type
country	C
region	C
state	C
city	C
zipcode	N

Target Column	Target Data	Type

1-1 Mappings	Targ
STAR.age --> age	HOL
STAR.brand --> brand	HOL
STAR.category --> category	HOL
STAR.city --> city	HOL
STAR.company --> company	HOL
STAR.costs --> costs	HOL
STAR.country --> country	HOL
STAR.cus_key --> cus_key	HOL
STAR.date --> date	HOL
STAR.delivery --> delivery	HOL
STAR.drop_dat --> drop_dat	HOL
STAR.drop_key --> drop_key	HOL
STAR.g_age --> g_age	HOL
STAR.g_income --> g_income	HOL

Load Step Processes

□ The source code is generated by SAS/Warehouse Administrator, rather than user-written, as shown in the MDDB Load Process Attributes window:



Select responsibility for creating the Source Code.

■ SAS/Warehouse Administrator Generated

☐ User Written

Source Code Library:

Catalog Entry Name: (catalog.entry.type)

For more information about Process Editor Jobs, see Chapter 13, "Maintaining Jobs," on page 251. For more information about processes, see Chapter 14, "Maintaining Processes," on page 281.

Example: Creating Summary Data for a MOLAP Application

Overview

MOLAP supports OLAP performed on a multidimensional database, such as a SAS MDDB. Using the **Sales Detail Grp** Detail Logical Table discussed in this chapter, consider the design of an OLAP application that calls for storage strategy of one SAS MDDB; the MOLAP application will store all the necessary crossings.

To produce the appropriate summary data for the MOLAP application:

1 Create an OLAP Group of type MOLAP in which you assign the group name and the group type of MOLAP.

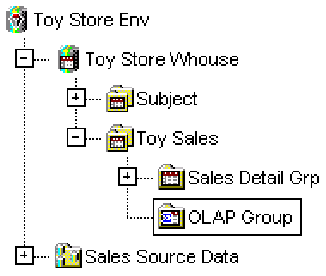
2 Create one OLAP MDDB in which you import all of the columns from the detail data source, assign the OLAP summary roles (class and statistic columns), define structure definitions (OLAP Cube, dimensions, and hierarchies), and define multiple crossings.

3 Define a Process Editor Job for the OLAP MDDB.

Note: The following explanations describe the metadata and methods used to achieve the desired results. It is assumed that the appropriate Data Warehouse Environment, Data Warehouse, Subject, and Detail Logical Table exist. △

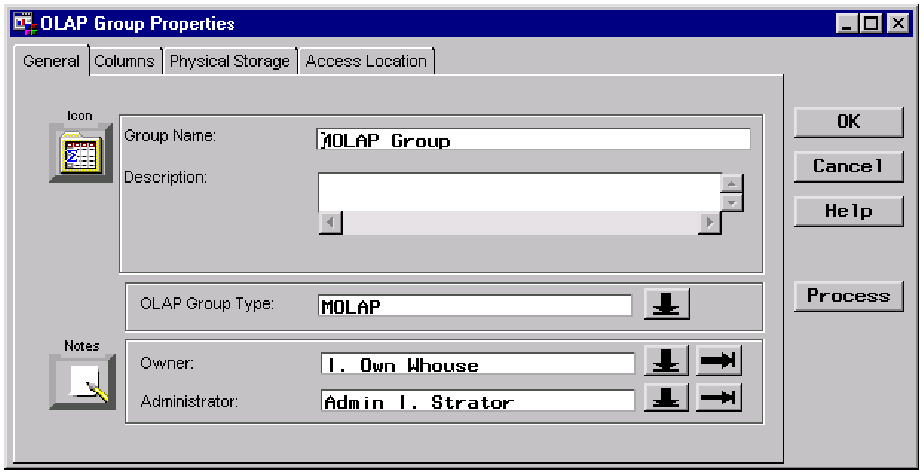
Define OLAP Group Properties

In the SAS/Warehouse Administrator Explorer, position the cursor on the Subject **Toy Sales**, click the right mouse button, select **Add Item**, and then **OLAP Group**. In the Explorer window, a new OLAP Group is added under the Subject as follows:



To update the default metadata for the OLAP Group, position the cursor on its icon, click the right mouse button, and select **`Properties`**. The OLAP Group Properties window displays for you to enter the appropriate information.

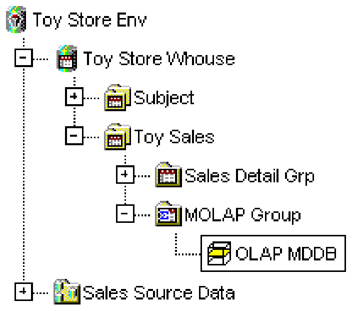
General Tab	specifies the group name **`MOLAP Group`**, description, group type **`MOLAP`**, owner, and administrator.



Group Name: MOLAP Group
Description:
OLAP Group Type: MOLAP
Owner: I. Own Whouse
Administrator: Admin I. Strator

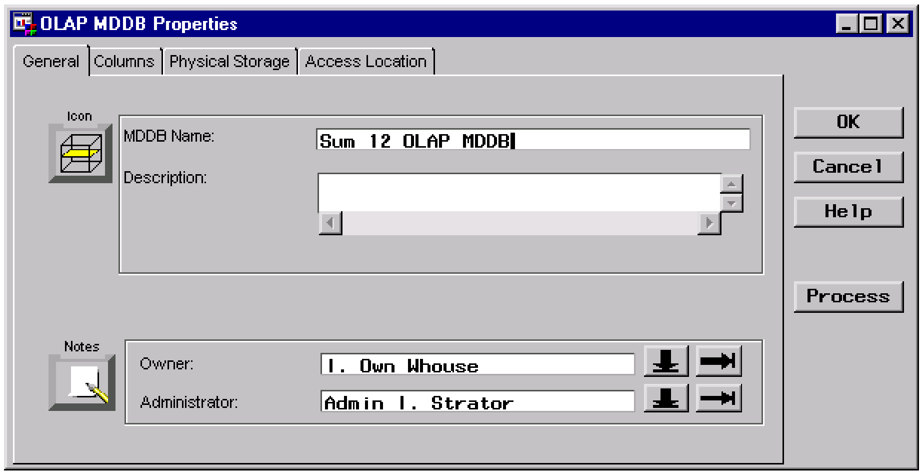
Define OLAP MDDB Properties

For the MOLAP application, the OLAP MDDB will store 12 crossings. In the SAS/Warehouse Administrator Explorer, position the cursor on the OLAP Group **`MOLAP Group`**, click the right mouse button, select **`Add Item`**, and then **`OLAP MDDB`**. In the Explorer window, a new OLAP MDDB is added under the OLAP Group as follows:



To update the default metadata for the OLAP MDDB Properties window, position the cursor on its icon, click the right mouse button, and select **Properties**. The OLAP MDDB Properties window displays for you to enter the appropriate information.

General Tab
: specifies the MDDB name **Sum 12 OLAP MDDB**, description, owner, and administrator.

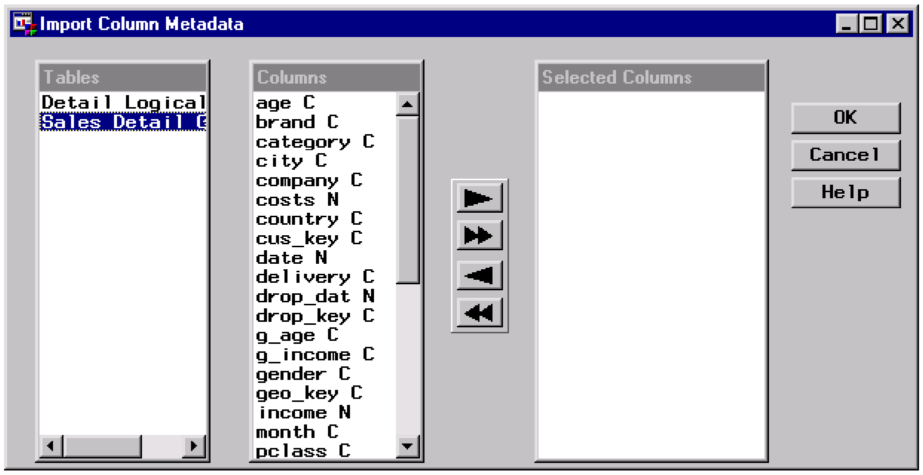


MDDB Name: Sum 12 OLAP MDDB
Description:
Owner: I. Own Whouse
Administrator: Admin I. Strator

Columns Tab
: specifies the columns to be included in the OLAP MDDB.

To import columns from an input source, click Import to display a list of input sources, and then select one (for example **Detail Logical Tables**). The Import Column Metadata window displays.

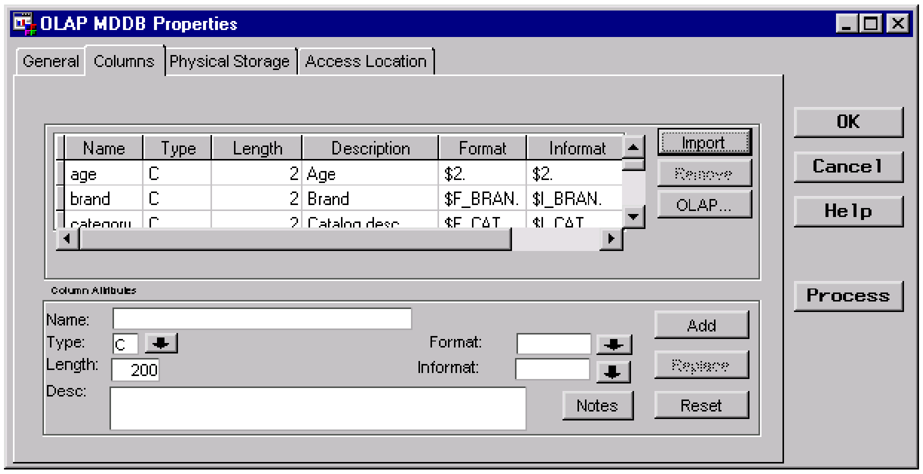
From the Import Column Metadata window, which lists the available input sources, select the Detail Logical Table **Sales Detail Grp** to display its available columns.



Tables:
Detail Logical
Sales Detail G

Columns:
age C
brand C
category C
city C
company C
costs N
country C
cus_key C
date N
delivery C
drop_dat N
drop_key C
g_age C
g_income C
gender C
geo_key C
income N
month C
pclass C

Select the appropriate columns listed under **Columns**, which for the OLAP MDDB are all 34 columns in the Detail Logical View. Use the double arrows to move them to **Selected Columns**, and then click OK. You are returned to the Columns tab in the OLAP MDDB Properties window, which lists the imported columns as follows:


OLAP MDDB Properties

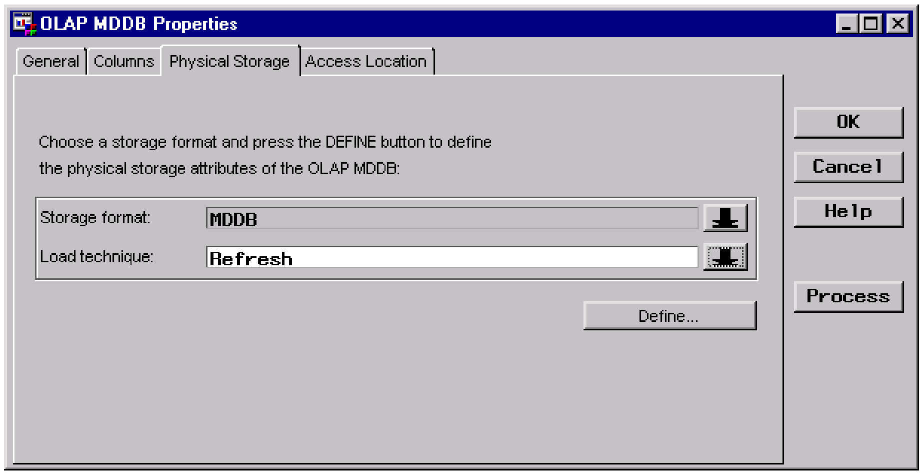
General | Columns | Physical Storage | Access Location

Name	Type	Length	Description	Format	Informat
age	C	2	Age	$2.	$2.
brand	C	2	Brand	$F_BRAN.	$I_BRAN.
category	C	2	Catalog desc	$F_CAT	$I_CAT

Column Attributes

Name:
Type: C
Length: 200
Desc:
Format:
Informat:

Physical Storage Tab specifies physical storage attributes. This example specifies the storage format **MDDB** and the load technique **Refresh**.


OLAP MDDB Properties

General | Columns | Physical Storage | Access Location

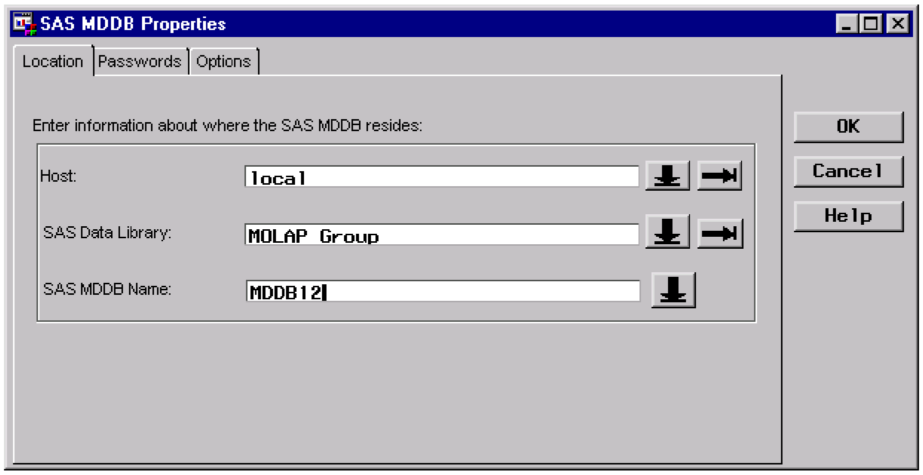
Choose a storage format and press the DEFINE button to define the physical storage attributes of the OLAP MDDB:

Storage format: MDDB

Load technique: Refresh

Click Define to open the SAS MDDB Properties window.

Location Tab specifies where the SAS MDDB is stored.


SAS MDDB Properties

Location | Passwords | Options

Enter information about where the SAS MDDB resides:

Host: local

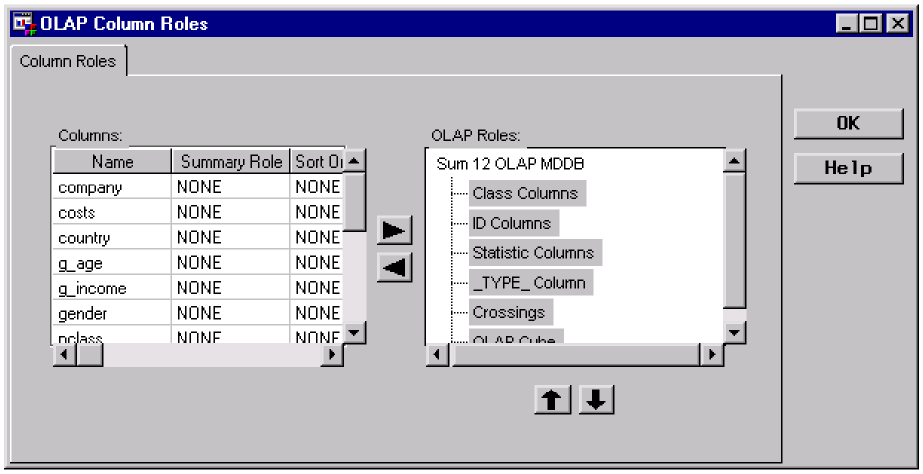
SAS Data Library: MOLAP Group

SAS MDDB Name: MDDB12

Assign OLAP Summary Roles for the MDDB

Assign Class Columns

From the Columns tab in the OLAP MDDB Properties window, click OLAP to open the OLAP Column Roles window from which you assign specific OLAP summary roles.



Columns:

Name	Summary Role	Sort O
company	NONE	NONE
costs	NONE	NONE
country	NONE	NONE
g_age	NONE	NONE
g_income	NONE	NONE
gender	NONE	NONE

OLAP Roles:

```
Sum 12 OLAP MDDB
  Class Columns
  ID Columns
  Statistic Columns
  _TYPE_ Column
  Crossings
```

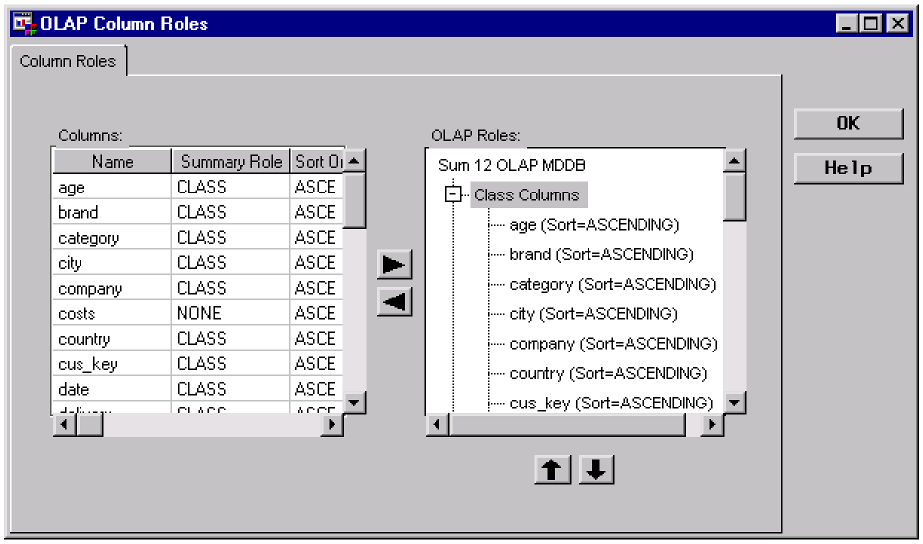
To assign class columns:

1 Select the columns from the list under **`Columns`**.

2 Select the **`Class Columns`** summary role label in the organization chart under **`OLAP Roles`**.

3 Click the right arrow to add the columns to the summary role.

For this example, class columns are assigned to all columns except three, which will be assigned as statistic columns. The default sort order ASCENDING is used for all class columns.



Columns:

Name	Summary Role	Sort O
age	CLASS	ASCE
brand	CLASS	ASCE
category	CLASS	ASCE
city	CLASS	ASCE
company	CLASS	ASCE
costs	NONE	ASCE
country	CLASS	ASCE
cus_key	CLASS	ASCE
date	CLASS	ASCE

OLAP Roles:

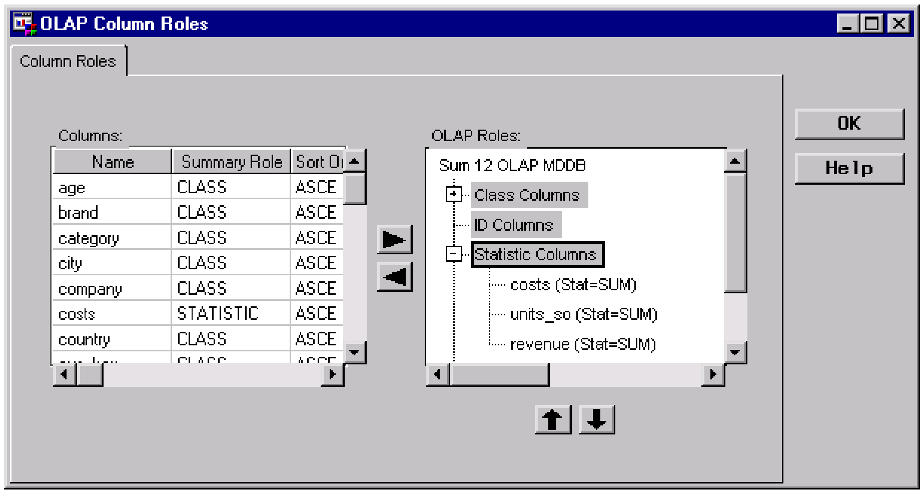
```
Sum 12 OLAP MDDB
  Class Columns
    age (Sort=ASCENDING)
    brand (Sort=ASCENDING)
    category (Sort=ASCENDING)
    city (Sort=ASCENDING)
    company (Sort=ASCENDING)
    country (Sort=ASCENDING)
    cus_key (Sort=ASCENDING)
```

Assign Statistic Columns

To assign statistic columns:

1 Select the columns from the list under **Columns**.
2 Select the **Statistic Columns** summary role label in the organization chart under **OLAP Roles**.
3 Click the right arrow to add the columns to the summary role.

For this example, statistic column is assigned to the remaining three columns, which are the numeric columns **costs**, **units_so**, and **revenue**. All three are assigned the default statistic SUM.

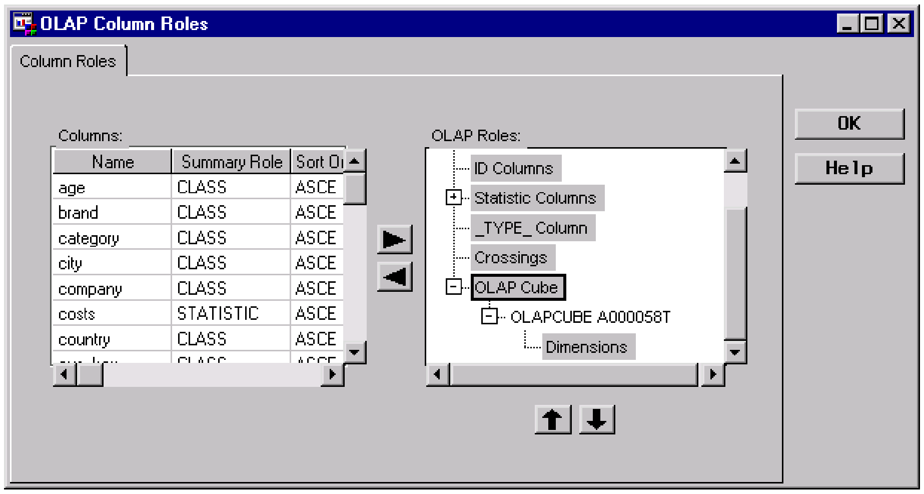


Name	Summary Role	Sort O
age	CLASS	ASCE
brand	CLASS	ASCE
category	CLASS	ASCE
city	CLASS	ASCE
company	CLASS	ASCE
costs	STATISTIC	ASCE
country	CLASS	ASCE

Define OLAP Structure Definitions for the MDDB

Add an OLAP Cube

From the OLAP Column Roles window, add an OLAP Cube by selecting the **OLAP Cube** label in the organization chart. Then click the right mouse button, and select **New**.



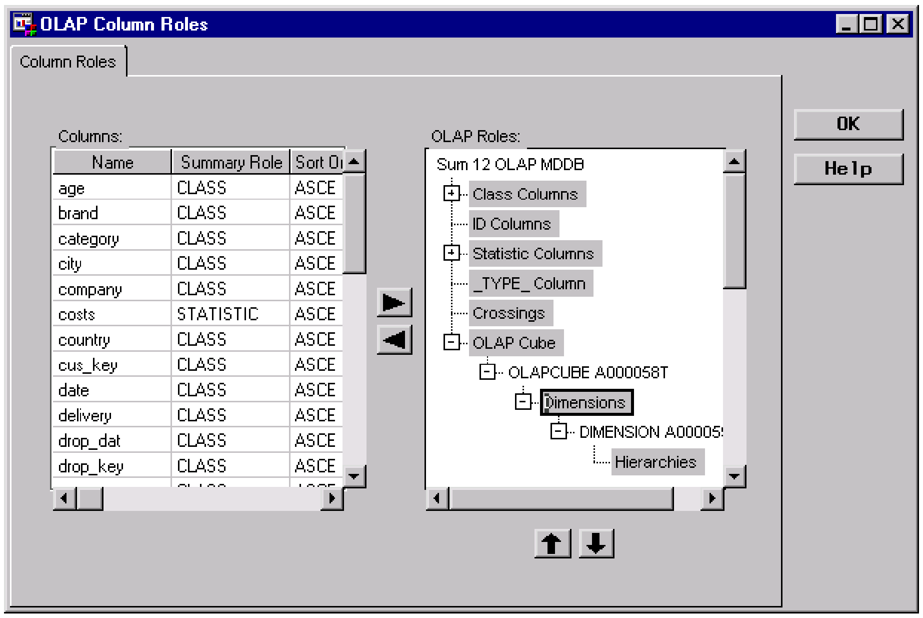
Name	Summary Role	Sort O
age	CLASS	ASCE
brand	CLASS	ASCE
category	CLASS	ASCE
city	CLASS	ASCE
company	CLASS	ASCE
costs	STATISTIC	ASCE
country	CLASS	ASCE

Define Dimensions

Dimension objects need to be defined for each anticipated dimension. As discussed in "Determining Dimensions in the Data" on page 180, this example defines the following dimensions:

- □ Geography
- □ Time
- □ Product
- □ CustomerGender
- □ CustomerAge
- □ CustomerIncome
- □ Drop
- □ Promotion

To define a dimension for the OLAP Cube, select the **`Dimensions`** label in the organization chart, click the right mouse button, and then select **`New`**.



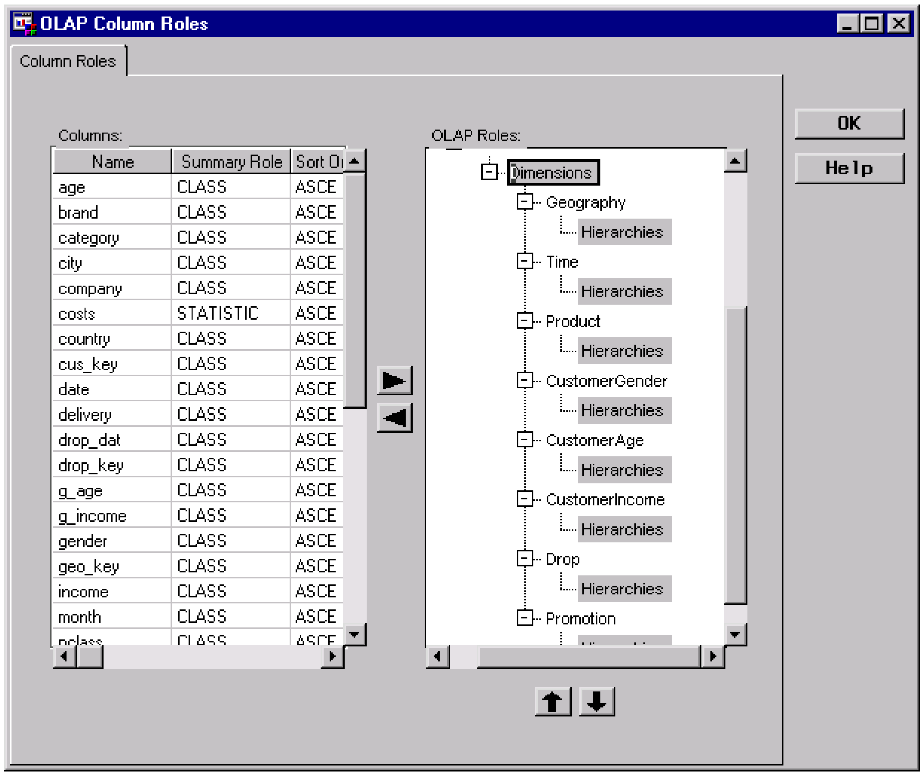
Columns:

Name	Summary Role	Sort Or
age	CLASS	ASCE
brand	CLASS	ASCE
category	CLASS	ASCE
city	CLASS	ASCE
company	CLASS	ASCE
costs	STATISTIC	ASCE
country	CLASS	ASCE
cus_key	CLASS	ASCE
date	CLASS	ASCE
delivery	CLASS	ASCE
drop_dat	CLASS	ASCE
drop_key	CLASS	ASCE

OLAP Roles:

```
Sum 12 OLAP MDDB
 ⊞ Class Columns
   ID Columns
 ⊞ Statistic Columns
   _TYPE_ Column
   Crossings
 ⊟ OLAP Cube
    ⊟ OLAPCUBE A000058T
       ⊟ Dimensions
          ⊟ DIMENSION A00005
               Hierarchies
```

To specify a name and a description for a dimension, click the right mouse button on the Dimension object, select **`Properties`**, and then update the default metadata. The next window displays the defined dimensions:



Columns:

Name	Summary Role	Sort O
age	CLASS	ASCE
brand	CLASS	ASCE
category	CLASS	ASCE
city	CLASS	ASCE
company	CLASS	ASCE
costs	STATISTIC	ASCE
country	CLASS	ASCE
cus_key	CLASS	ASCE
date	CLASS	ASCE
delivery	CLASS	ASCE
drop_dat	CLASS	ASCE
drop_key	CLASS	ASCE
g_age	CLASS	ASCE
g_income	CLASS	ASCE
gender	CLASS	ASCE
geo_key	CLASS	ASCE
income	CLASS	ASCE
month	CLASS	ASCE

OLAP Roles:

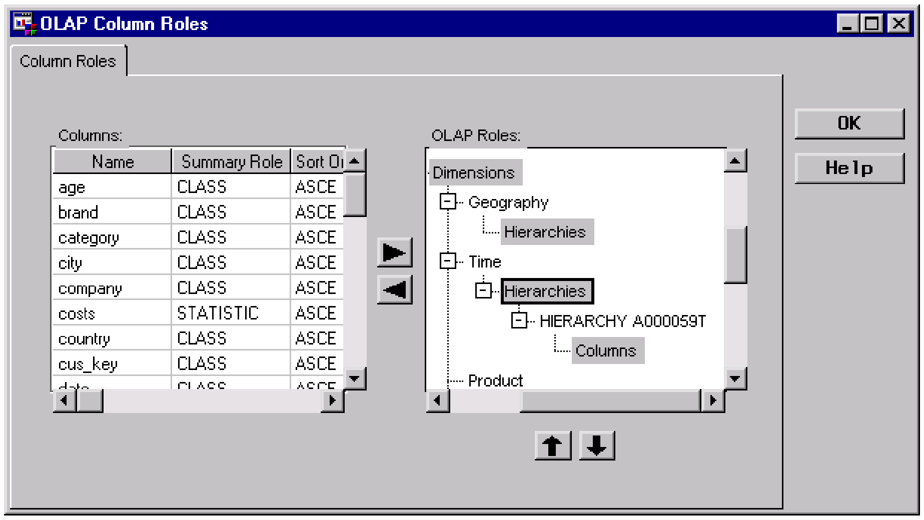
- Dimensions
  - Geography
    - Hierarchies
  - Time
    - Hierarchies
  - Product
    - Hierarchies
  - CustomerGender
    - Hierarchies
  - CustomerAge
    - Hierarchies
  - CustomerIncome
    - Hierarchies
  - Drop
    - Hierarchies
  - Promotion

Define Hierarchies

Hierarchies are defined for each anticipated hierarchy, which is a member of a dimension. For example, the **Time** dimension will include these hierarchies:

- ☐ TimeWeek
- ☐ TimeQuarter
- ☐ TimeSeason

To define a hierarchy for a dimension, select the **Hierarchies** label in the organization chart associated with the appropriate dimension (for example, **Time**) click the right mouse button, and then select **New**.



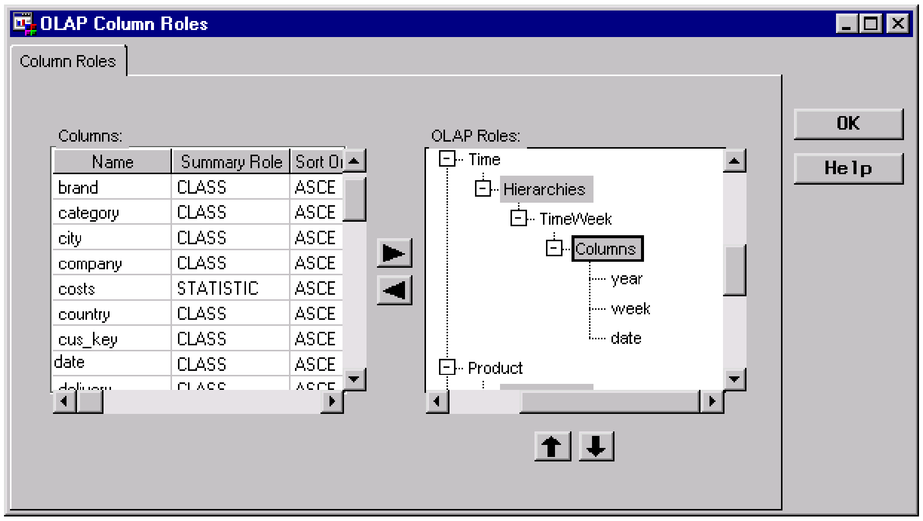
Columns:

Name	Summary Role	Sort O
age	CLASS	ASCE
brand	CLASS	ASCE
category	CLASS	ASCE
city	CLASS	ASCE
company	CLASS	ASCE
costs	STATISTIC	ASCE
country	CLASS	ASCE
cus_key	CLASS	ASCE

OLAP Roles:

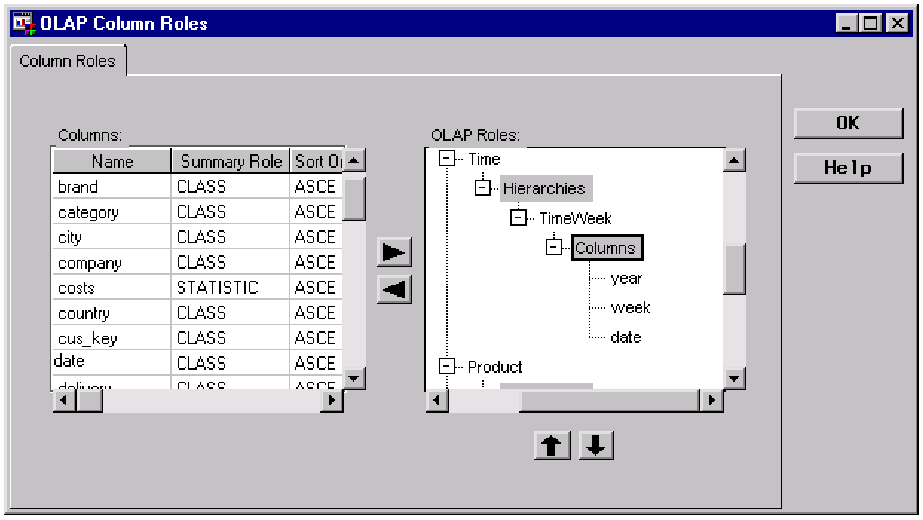
- Dimensions
  - Geography
    - Hierarchies
  - Time
    - Hierarchies
      - HIERARCHY A000059T
        - Columns
  - Product

To specify a name (for example, **TimeWeek**) and a description, click the right mouse button on the Hierarchy object, select **Properties**, and then update the default metadata.



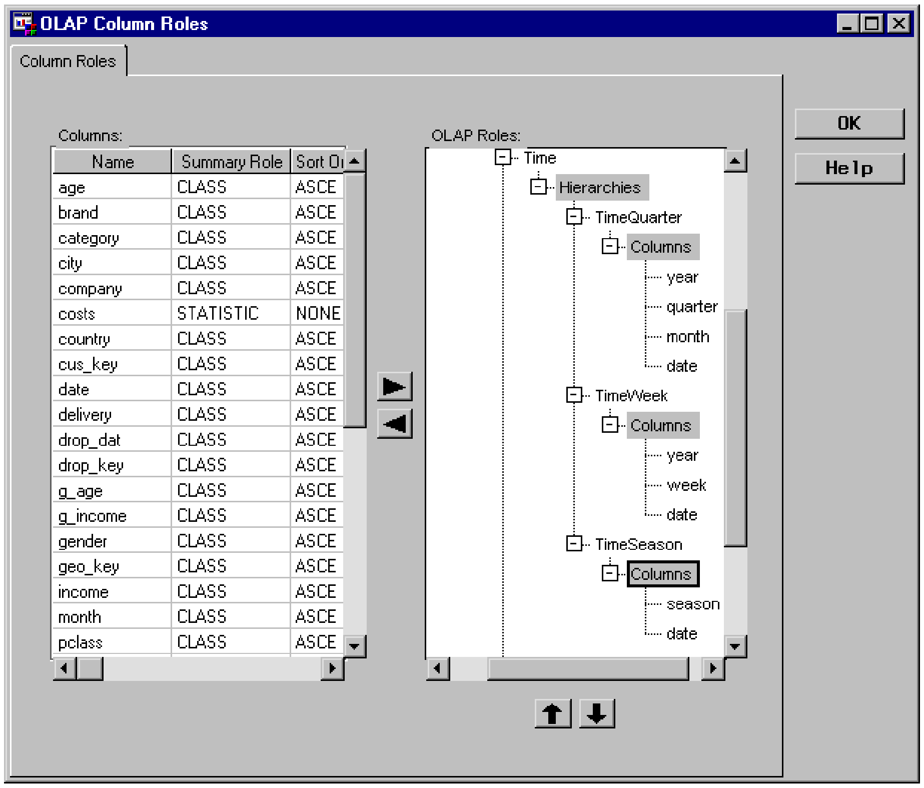
Name	Summary Role	Sort O
brand	CLASS	ASCE
category	CLASS	ASCE
city	CLASS	ASCE
company	CLASS	ASCE
costs	STATISTIC	ASCE
country	CLASS	ASCE
cus_key	CLASS	ASCE
date	CLASS	ASCE

Then, to specify the appropriate column(s) for the hierarchy (such as **year**, **week**, and **date**) select the class columns from the list under **Columns**, select the **Columns** label associated with the hierarchy object in the organization chart, and click the right arrow to add the columns to the object.



Name	Summary Role	Sort O
brand	CLASS	ASCE
category	CLASS	ASCE
city	CLASS	ASCE
company	CLASS	ASCE
costs	STATISTIC	ASCE
country	CLASS	ASCE
cus_key	CLASS	ASCE
date	CLASS	ASCE

The following window displays the defined hierarchies with appropriate columns for the **Time** dimension:



Columns:

Name	Summary Role	Sort O
age	CLASS	ASCE
brand	CLASS	ASCE
category	CLASS	ASCE
city	CLASS	ASCE
company	CLASS	ASCE
costs	STATISTIC	NONE
country	CLASS	ASCE
cus_key	CLASS	ASCE
date	CLASS	ASCE
delivery	CLASS	ASCE
drop_dat	CLASS	ASCE
drop_key	CLASS	ASCE
g_age	CLASS	ASCE
g_income	CLASS	ASCE
gender	CLASS	ASCE
geo_key	CLASS	ASCE
income	CLASS	ASCE
month	CLASS	ASCE
pclass	CLASS	ASCE

OLAP Roles:

```
Time
  Hierarchies
    TimeQuarter
      Columns
        year
        quarter
        month
        date
    TimeWeek
      Columns
        year
        week
        date
    TimeSeason
      Columns
        season
        date
```

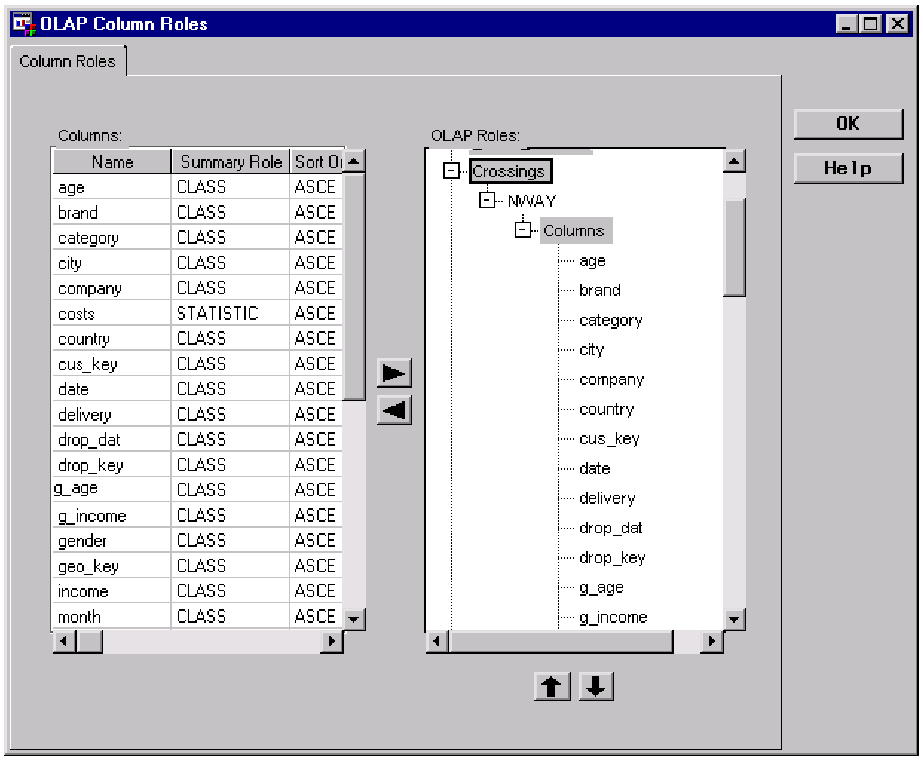
Define Crossings for the MDDB

For the MOLAP application, the OLAP MDDB will store the following crossings:

- □ an NWAY crossing, which consists of all the assigned class columns
- □ one crossing for the 12 most likely accessed class columns
- □ 11 crossings, with each one having one less column than the crossing with the 12 class columns.

Note: For a MOLAP application, you must create an NWAY crossing for all the class columns defined for the MDDB, which consists of 31 columns. Even though the detail data is available from a permanent data store, a MOLAP application (unlike a HOLAP application) cannot access all the detail data without it being referenced in a crossing. △

To define the NWAY crossing, in the organization chart on the **`Crossings`** label, click the right mouse button, and select **`Create NWAY Crossing`**. The resulting NWAY crossing follows:



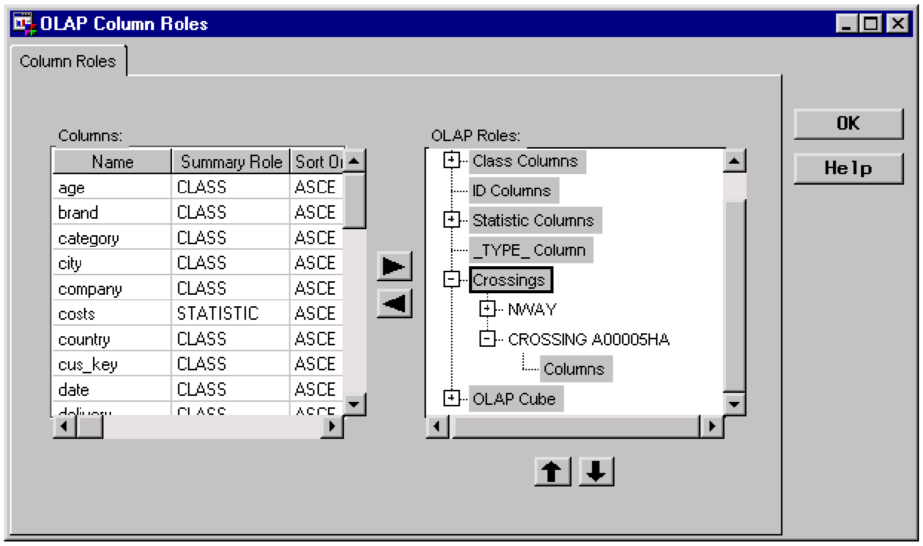
Columns:

Name	Summary Role	Sort O
age	CLASS	ASCE
brand	CLASS	ASCE
category	CLASS	ASCE
city	CLASS	ASCE
company	CLASS	ASCE
costs	STATISTIC	ASCE
country	CLASS	ASCE
cus_key	CLASS	ASCE
date	CLASS	ASCE
delivery	CLASS	ASCE
drop_dat	CLASS	ASCE
drop_key	CLASS	ASCE
g_age	CLASS	ASCE
g_income	CLASS	ASCE
gender	CLASS	ASCE
geo_key	CLASS	ASCE
income	CLASS	ASCE
month	CLASS	ASCE

OLAP Roles:

```
Crossings
    NWAY
        Columns
            age
            brand
            category
            city
            company
            country
            cus_key
            date
            delivery
            drop_dat
            drop_key
            g_age
            g_income
```

To create the crossing to represent the 12 most likely accessed class columns, in the organization chart on the **Crossings** label, click the right mouse button and select **New**. For an explanation about how these 12 class columns were determined, see “Determining Crossing(s)” on page 181. The new crossing is added under the **Crossings** label as follows:



Columns:

Name	Summary Role	Sort O
age	CLASS	ASCE
brand	CLASS	ASCE
category	CLASS	ASCE
city	CLASS	ASCE
company	CLASS	ASCE
costs	STATISTIC	ASCE
country	CLASS	ASCE
cus_key	CLASS	ASCE
date	CLASS	ASCE

OLAP Roles:

```
Class Columns
ID Columns
Statistic Columns
_TYPE_ Column
Crossings
    NWAY
    CROSSING A00005HA
        Columns
OLAP Cube
```

To assign a name to the crossing, click the right mouse button on the crossing and select **Properties**.

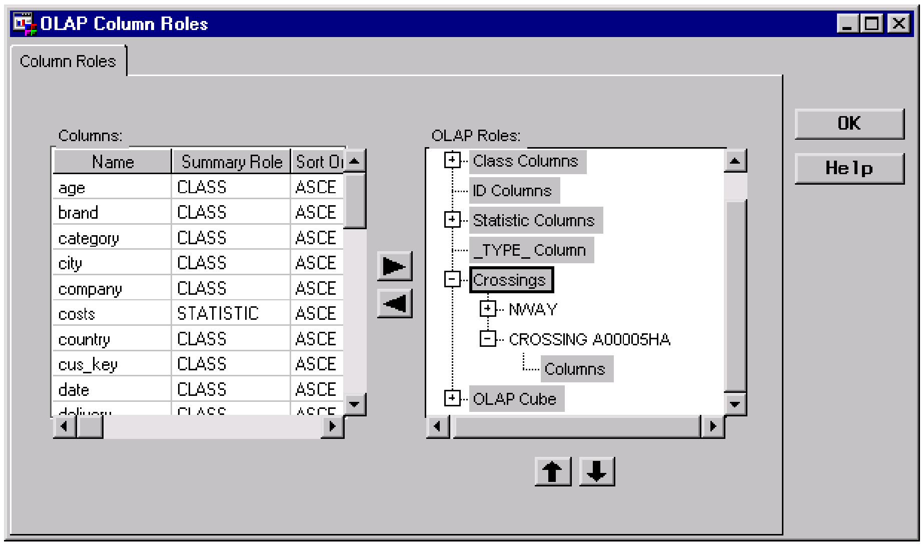
To specify the class columns for the crossing:

1 Select the columns from the **Columns** list.

2 Select the **Columns** label in the organization chart under the added crossing.

3 Click the right arrow to add the columns to the crossing.

The following window displays the added crossing with the name **`12 Columns Crossing`** and the added columns:



OLAP Column Roles

Column Roles

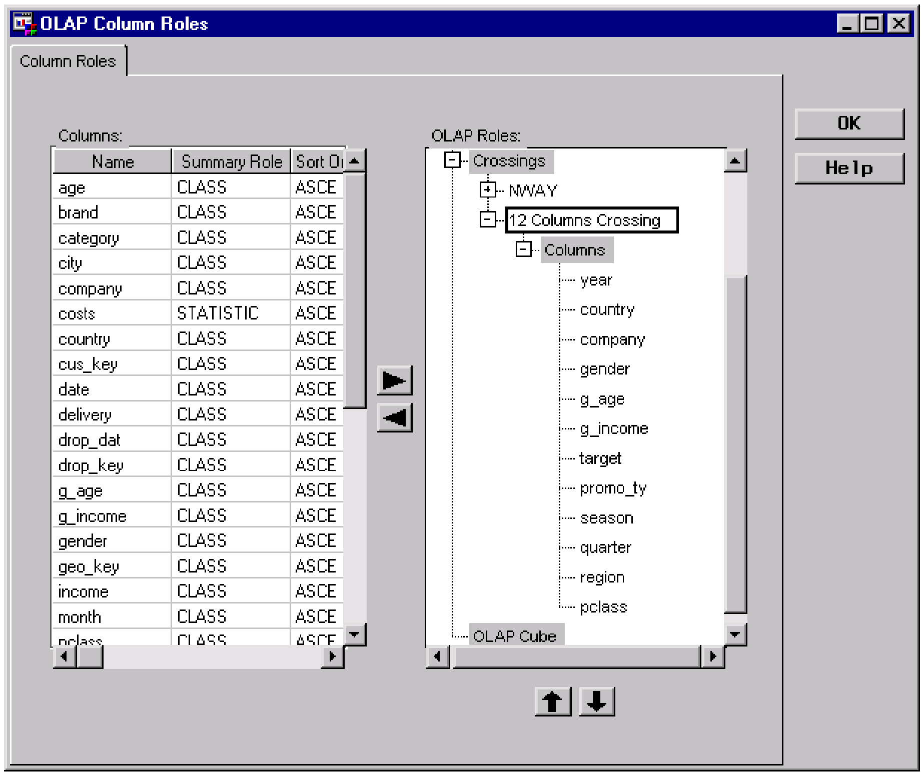
Columns:

Name	Summary Role	Sort O
age	CLASS	ASCE
brand	CLASS	ASCE
category	CLASS	ASCE
city	CLASS	ASCE
company	CLASS	ASCE
costs	STATISTIC	ASCE
country	CLASS	ASCE
cus_key	CLASS	ASCE
date	CLASS	ASCE

OLAP Roles:

```
Class Columns
ID Columns
Statistic Columns
_TYPE_ Column
Crossings
    NWAY
    CROSSING A00005HA
        Columns
OLAP Cube
```

To create the remaining 11 crossings, in the organization chart on the crossing **`12 Columns Crossing`**, click the right mouse button, and select **`Create Stairstep Crossings`**. Eleven crossings are added, with each one having one less column than the crossing before it.



OLAP Column Roles

Column Roles

Columns:

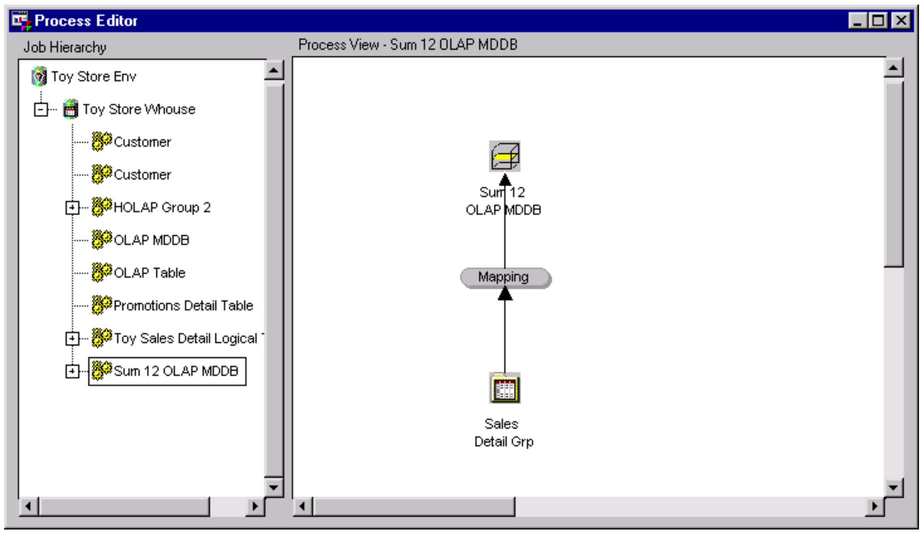
Name	Summary Role	Sort O
age	CLASS	ASCE
brand	CLASS	ASCE
category	CLASS	ASCE
city	CLASS	ASCE
company	CLASS	ASCE
costs	STATISTIC	ASCE
country	CLASS	ASCE
cus_key	CLASS	ASCE
date	CLASS	ASCE
delivery	CLASS	ASCE
drop_dat	CLASS	ASCE
drop_key	CLASS	ASCE
g_age	CLASS	ASCE
g_income	CLASS	ASCE
gender	CLASS	ASCE
geo_key	CLASS	ASCE
income	CLASS	ASCE
month	CLASS	ASCE

OLAP Roles:

```
Crossings
    NWAY
    12 Columns Crossing
        Columns
            year
            country
            company
            gender
            g_age
            g_income
            target
            promo_ty
            season
            quarter
            region
            pclass
OLAP Cube
```

Define Process Editor Job

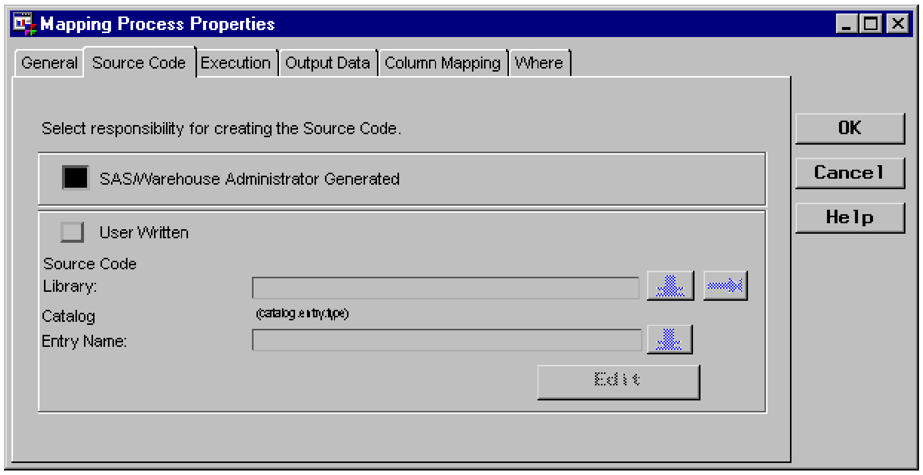
In the Process Editor Job, the OLAP MDDB **`SUM 12 OLAP MDDB`** is specified as the output target. The Detail Logical Table **`Sales Detail Grp`** is specified as the input source. The following Process Editor window displays the Process Flow for the Job:



The processes defined in the Job are summarized as follows:

Mapping Process

- ☐ The source code to map columns is generated by SAS/Warehouse Administrator, rather than user-written, as shown in the Source Code tab:



Select responsibility for creating the Source Code.

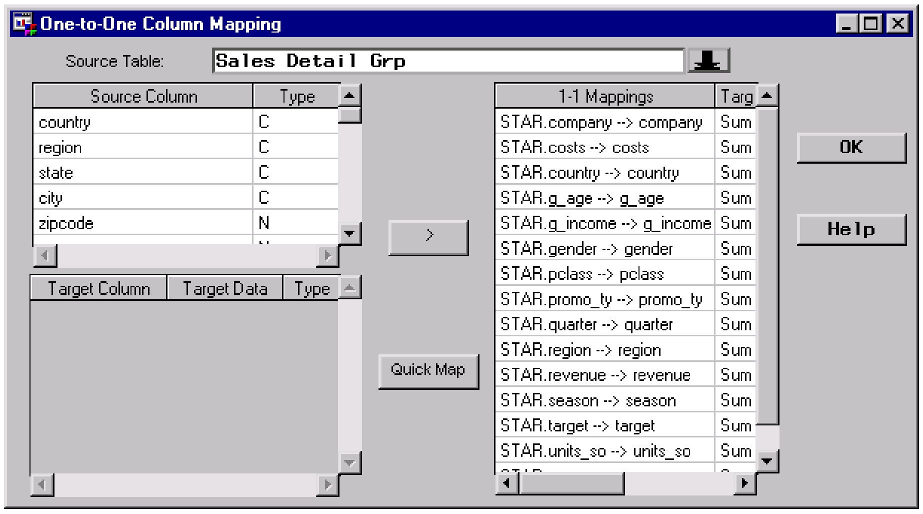
■ SAS/Warehouse Administrator Generated

☐ User Written

Source Code Library:

Catalog Entry Name: (catalog.entry.type)

- ☐ Column mapping is defined as one-to-one mapping. To produce one-to-one mapping, first click $\boxed{\text{1 to 1 Mappings}}$ on the Column Mapping tab in the Mapping Process Properties window, which opens the One-to-One Column Mapping window. Then, click $\boxed{\text{Quick Map}}$, which produces the following results:



Source Table: Sales Detail Grp

Source Column	Type
country	C
region	C
state	C
city	C
zipcode	N

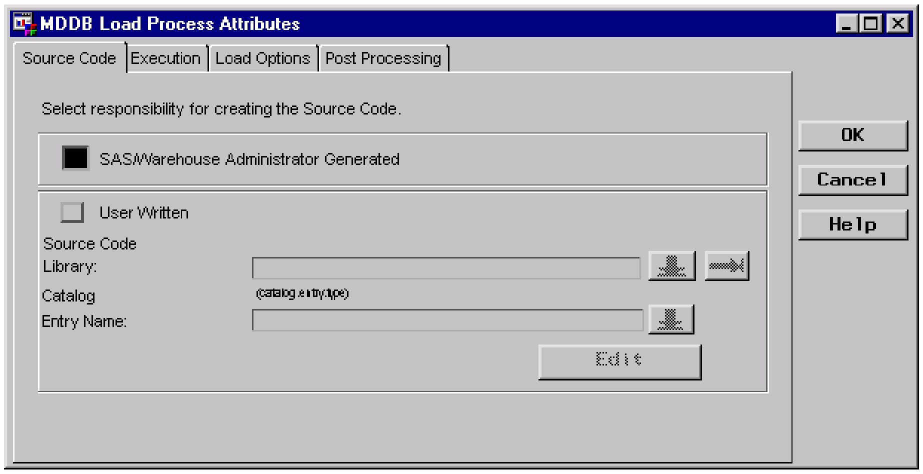
Target Column	Target Data	Type

Quick Map

1-1 Mappings	Targ
STAR.company --> company	Sum
STAR.costs --> costs	Sum
STAR.country --> country	Sum
STAR.g_age --> g_age	Sum
STAR.g_income --> g_income	Sum
STAR.gender --> gender	Sum
STAR.pclass --> pclass	Sum
STAR.promo_ty --> promo_ty	Sum
STAR.quarter --> quarter	Sum
STAR.region --> region	Sum
STAR.revenue --> revenue	Sum
STAR.season --> season	Sum
STAR.target --> target	Sum
STAR.units_so --> units_so	Sum

Load Step Process

□ The source code is generated by SAS/Warehouse Administrator, rather than user-written, as shown in the MDDB Load Process Attributes window:



Select responsibility for creating the Source Code.

■ SAS/Warehouse Administrator Generated

□ User Written

Source Code Library:

Catalog (catalog.entry.type)

Entry Name:

For more information about Process Editor Jobs, see Chapter 13, "Maintaining Jobs," on page 251. For more information about processes, see Chapter 14, "Maintaining Processes," on page 281.

Example: Adding a Frequency Count to an OLAP Summary Data Store

Overview

For an OLAP summary data store, you might need the summary data to contain the frequency of each record. That is, you want a numeric column whose value represents the frequency of the rows for each combination of class column values. To produce the appropriate summary data:

1 Add a column to the summary data store.

2 Assign the added column as a statistic column, using the SUM statistic.

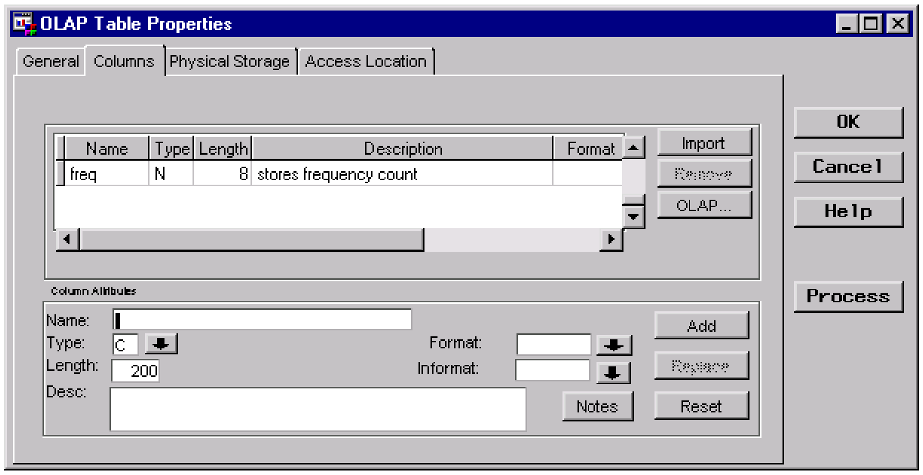
3 In the Mapping process for the summary data store, map the column to a value of `1`.

Note: The following explanations describe only the metadata and methods that specifically apply to this example. △

Add a Column to OLAP Summary Data Store

This explanation adds a column to an OLAP Table. You add a column from the Columns tab in the object's properties window, which for this example is the OLAP Table Properties window.

Columns Tab specifies the columns to be included in the summary data store. To add a column that does not exist in the input data, define the column's attributes (name, type, length, description, format, and informat), then click Add. The result of adding a column to store frequency count follows:



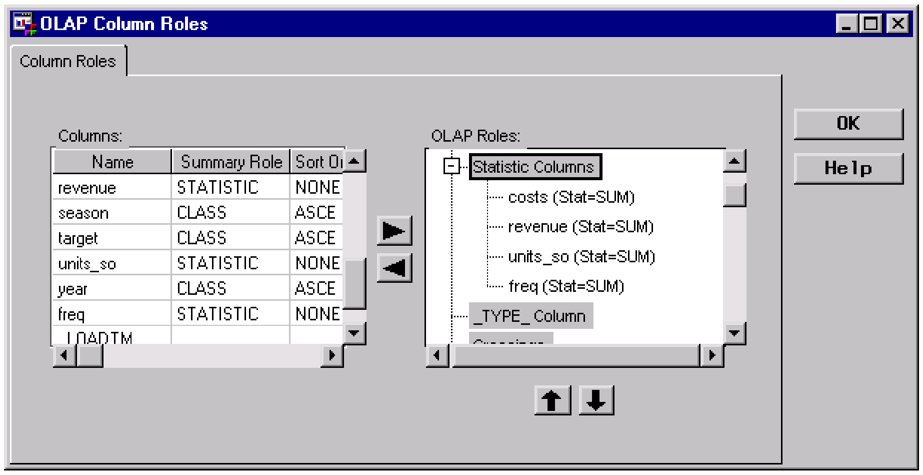
Name	Type	Length	Description	Format
freq	N	8	stores frequency count	

Name:
Type: C
Length: 200
Desc:
Format:
Informat:

Assign OLAP Summary Roles to the Column

On the Columns tab, click OLAP to open the OLAP Column Roles window from which you assign specific OLAP summary roles.

Assign the **`freq`** column as a statistic column, using the SUM statistic, as follows:

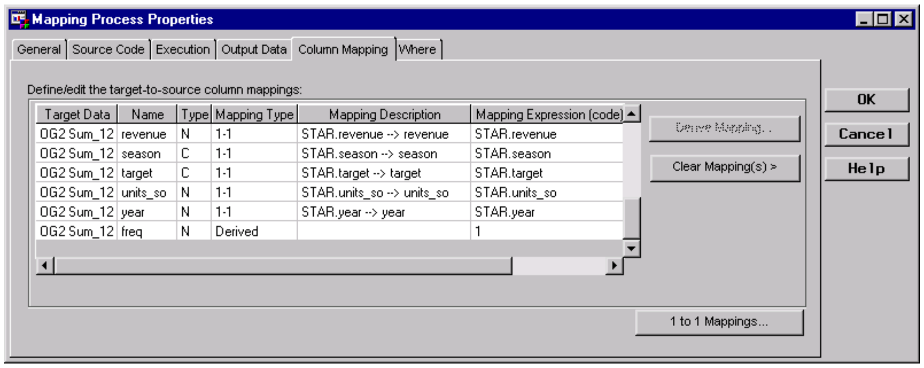


Name	Summary Role	Sort O
revenue	STATISTIC	NONE
season	CLASS	ASCE
target	CLASS	ASCE
units_so	STATISTIC	NONE
year	CLASS	ASCE
freq	STATISTIC	NONE
LOADTM		

```
Statistic Columns
    costs (Stat=SUM)
    revenue (Stat=SUM)
    units_so (Stat=SUM)
    freq (Stat=SUM)
_TYPE_Column
```

Define Mapping Process in Process Editor Job

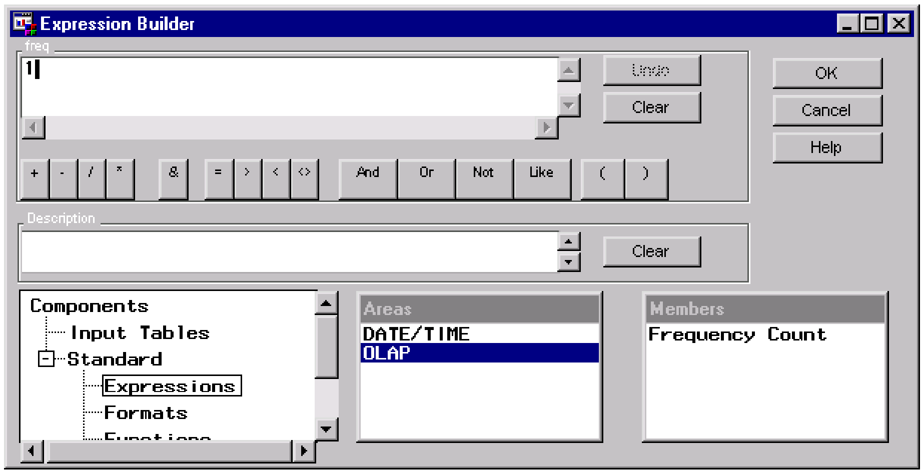
The **`freq`** column must be mapped to a value of **`1`** as shown in the following Mapping Process Properties window:



Define/edit the target-to-source column mappings:

Target Data	Name	Type	Mapping Type	Mapping Description	Mapping Expression (code)
OG2 Sum_12	revenue	N	1-1	STAR.revenue --> revenue	STAR.revenue
OG2 Sum_12	season	C	1-1	STAR.season --> season	STAR.season
OG2 Sum_12	target	C	1-1	STAR.target --> target	STAR.target
OG2 Sum_12	units_so	N	1-1	STAR.units_so --> units_so	STAR.units_so
OG2 Sum_12	year	N	1-1	STAR.year --> year	STAR.year
OG2 Sum_12	freq	N	Derived		1

You can do this easily by using the stored expression **`Frequency Count`**, which is available in the Expression Builder window. That is, from the Column Mapping tab on the Mapping Process Properties window, select the column, then click Derived Mapping, which opens the Expression Builder window. From the Expression Builder window, select the **`Expressions`** component **`OLAP`**, and then **`Frequency Count`**.



freq: 1

Components	Areas	Members
Input Tables	DATE/TIME	Frequency Count
Standard	OLAP	
Expressions		
Formats		
Functions		

When you click OK on the Expression Builder window, the value of **`1`** is assigned to the **`freq`** column, and you are returned to the Column Mapping tab in the Mapping Process Properties window.

Example: Using One Analysis Column for Multiple Statistic Columns

Overview

Suppose you want to use a specific column from input data to generate multiple statistics for your summary data. That is, you want to use one input analysis column for multiple output statistic columns. To produce the appropriate statistics:

1 Add the additional column(s) to the summary data store.

2 Assign each added column as a statistic column, using the appropriate statistic.

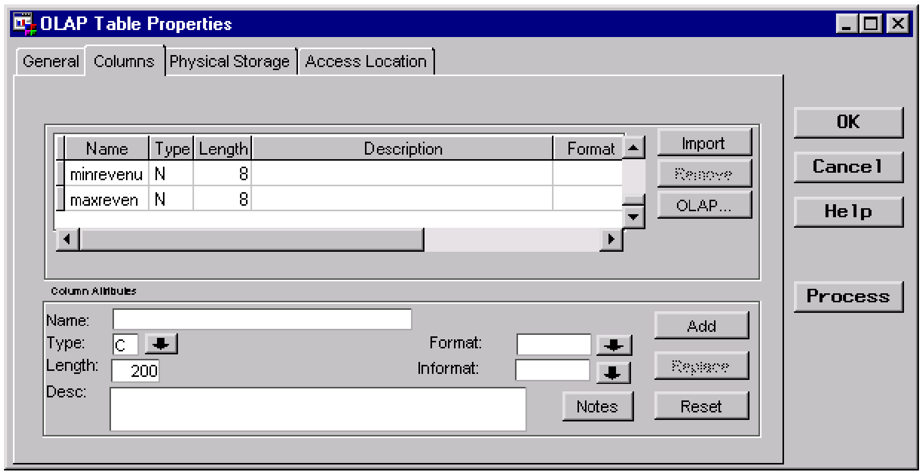
3 In the Mapping process for the summary data store, map the added column(s) to the analysis column.

Note: The following explanations describe only the metadata and methods that specifically apply to this example. △

Add Columns to OLAP Summary Data Store

This explanation adds two numeric columns named **`minrevenue`** and **`maxrevenue`** to an OLAP Table. You add columns from the Columns tab in the object's properties window, which for this example is the OLAP Table Properties window.

Columns Tab specifies the columns to be included in the summary data store. To add a column that does not exist in the input data, define the column's attributes (name, type, length, description, format, and informat), and then click Add. The result of adding two columns to store generated statistics is as follows:



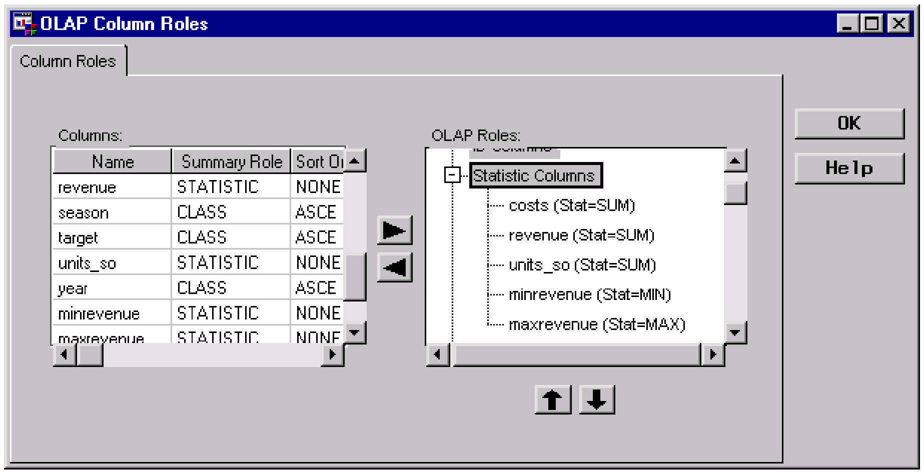
Name	Type	Length	Description	Format
minrevenu	N	8		
maxreven	N	8		

Name:
Type: C
Length: 200
Desc:
Format:
Informat:

Assign OLAP Summary Roles to the Columns

On the Columns tab, click OLAP to open the OLAP Column Roles window from which you assign specific OLAP summary roles.

Assign the **`minrevenue`** and **`maxrevenue`** columns as statistic columns, using the statistics MIN and MAX, as follows:



OLAP Column Roles

Column Roles

Columns:

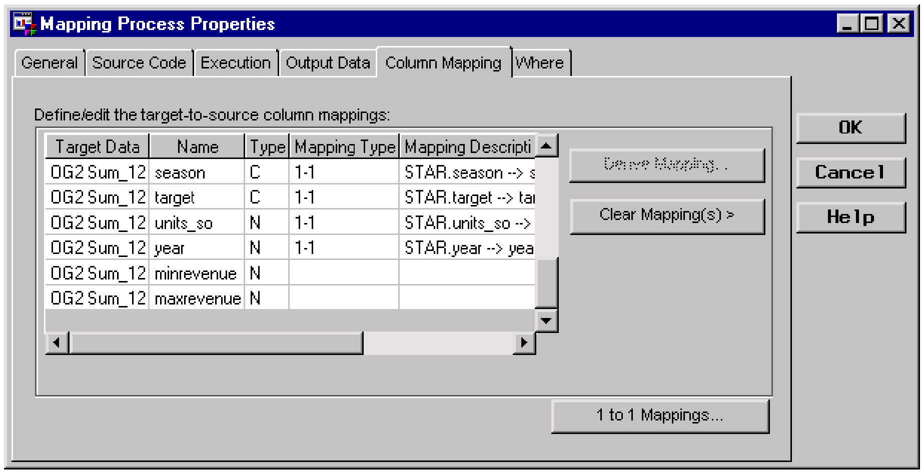
Name	Summary Role	Sort O
revenue	STATISTIC	NONE
season	CLASS	ASCE
target	CLASS	ASCE
units_so	STATISTIC	NONE
year	CLASS	ASCE
minrevenue	STATISTIC	NONE
maxrevenue	STATISTIC	NONE

OLAP Roles:

Statistic Columns
- costs (Stat=SUM)
- revenue (Stat=SUM)
- units_so (Stat=SUM)
- minrevenue (Stat=MIN)
- maxrevenue (Stat=MAX)

Define Mapping Process in Process Editor Job

The added statistic columns, shown in the following Mapping Process Properties window, must be mapped to the analysis column.



Mapping Process Properties

General | Source Code | Execution | Output Data | Column Mapping | Where

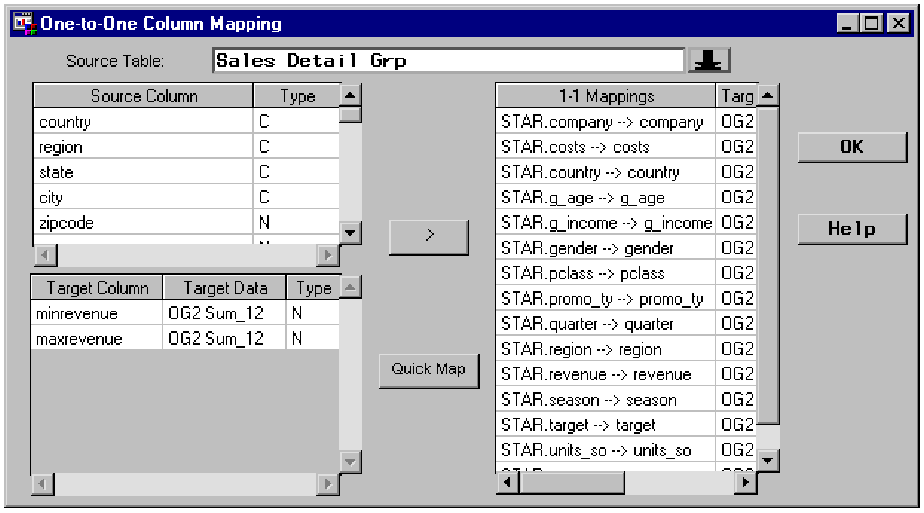
Define/edit the target-to-source column mappings:

Target Data	Name	Type	Mapping Type	Mapping Descripti
OG2 Sum_12	season	C	1-1	STAR.season --> s
OG2 Sum_12	target	C	1-1	STAR.target --> ta
OG2 Sum_12	units_so	N	1-1	STAR.units_so -->
OG2 Sum_12	year	N	1-1	STAR.year --> yea
OG2 Sum_12	minrevenue	N		
OG2 Sum_12	maxrevenue	N		

To map the added statistic columns to the analysis column:

1 From the Column Mapping tab in the Mapping Process Properties window, select the column, such as **`minrevenue`**.

2 Click 1-to-1 Mappings to open the One-to-One Column Mapping window as follows:



Source Column	Type
country	C
region	C
state	C
city	C
zipcode	N

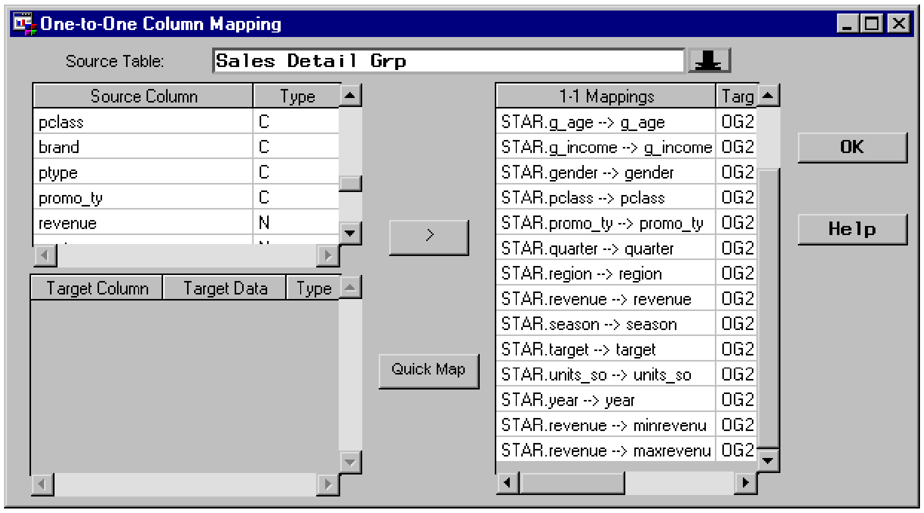
Target Column	Target Data	Type
minrevenue	OG2 Sum_12	N
maxrevenue	OG2 Sum_12	N

1-1 Mappings	Targ
STAR.company --> company	OG2
STAR.costs --> costs	OG2
STAR.country --> country	OG2
STAR.g_age --> g_age	OG2
STAR.g_income --> g_income	OG2
STAR.gender --> gender	OG2
STAR.pclass --> pclass	OG2
STAR.promo_ty --> promo_ty	OG2
STAR.quarter --> quarter	OG2
STAR.region --> region	OG2
STAR.revenue --> revenue	OG2
STAR.season --> season	OG2
STAR.target --> target	OG2
STAR.units_so --> units_so	OG2

3 Select the analysis column from under the label **`Source Column`**.

4 Select the statistic column from under the label **`Target Column`**.

5 Click the right arrow to map the two columns.

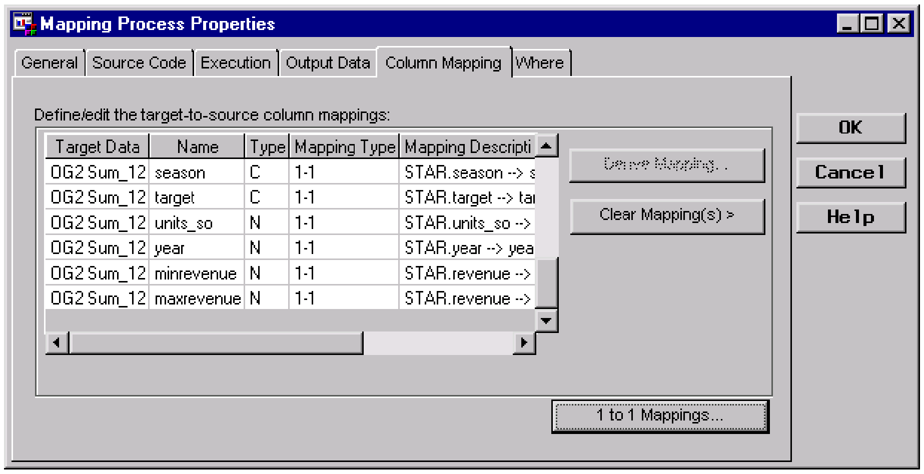


Source Column	Type
pclass	C
brand	C
ptype	C
promo_ty	C
revenue	N

Target Column	Target Data	Type

1-1 Mappings	Targ
STAR.g_age --> g_age	OG2
STAR.g_income --> g_income	OG2
STAR.gender --> gender	OG2
STAR.pclass --> pclass	OG2
STAR.promo_ty --> promo_ty	OG2
STAR.quarter --> quarter	OG2
STAR.region --> region	OG2
STAR.revenue --> revenue	OG2
STAR.season --> season	OG2
STAR.target --> target	OG2
STAR.units_so --> units_so	OG2
STAR.year --> year	OG2
STAR.revenue --> minrevenu	OG2
STAR.revenue --> maxrevenu	OG2

When you click $\boxed{\text{OK}}$, you are returned to the Column Mapping tab in the Mapping Process Properties window as follows:



Define/edit the target-to-source column mappings:

Target Data	Name	Type	Mapping Type	Mapping Descripti
OG2 Sum_12	season	C	1-1	STAR.season --> s
OG2 Sum_12	target	C	1-1	STAR.target --> ta
OG2 Sum_12	units_so	N	1-1	STAR.units_so -->
OG2 Sum_12	year	N	1-1	STAR.year --> yea
OG2 Sum_12	minrevenue	N	1-1	STAR.revenue -->
OG2 Sum_12	maxrevenue	N	1-1	STAR.revenue -->

Example: Using DATE/TIME Stored Expression to Split Date Values

Overview

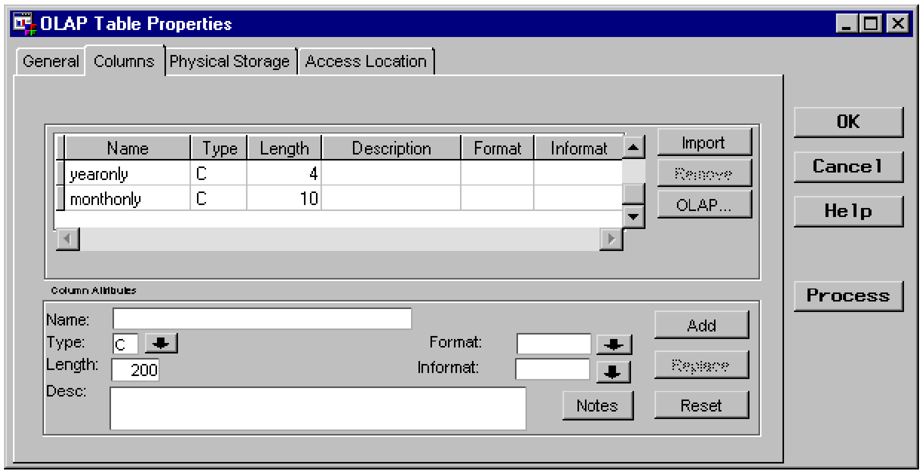
For an OLAP summary data store, you might need to store parts of date values from an input column. For example, suppose an input column named **`drop_dat`** contains SAS date values that correspond to a value **`05/05/95`**. For the data store, you could add columns to represent different parts of that value (for example, **`yearonly`** and **`monthonly`**) and split the value among the two output columns. To produce the appropriate data:

1. Add the columns **`yearonly`** and **`monthonly`** to the OLAP data store.
2. Assign the added columns as class columns.
3. For each added column, in the Mapping process, define a derived mapping to return a value using a specified format. The SAS/Warehouse Administrator Expression Builder window provides stored expressions from the **`DATE/TIME`** standard component so that you can easily define the derived mappings.

Add Columns to OLAP Summary Data Store

You add columns from the Columns tab in the object's properties window, which for this example is the OLAP Table Properties window. This example assumes that other columns have been imported from the input source and only illustrates adding columns to the data store.

Columns Tab	specifies the columns to be included in the OLAP data store. To add a column that does not exist in the input data, define the column's attributes (name, type, length, description, format, and informat), and then click Add. The result of adding columns to store the date values is as follows:



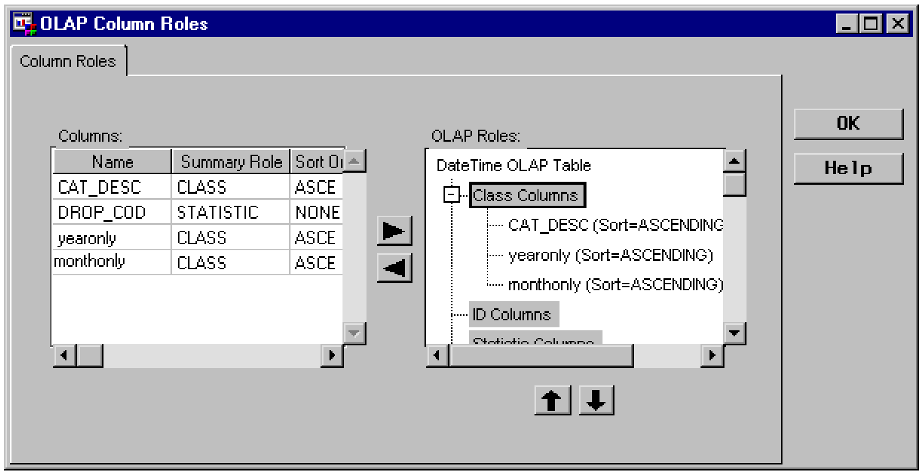
Name	Type	Length	Description	Format	Informat
yearonly	C	4			
monthonly	C	10			

Type: C
Length: 200

Assign OLAP Summary Roles to the Columns

On the Columns tab, click OLAP to open the OLAP Column Roles window from which you assign OLAP summary roles. This example assumes that the summary roles for the imported columns have been assigned.

For the added columns, assign **yearonly** and **monthonly** as class columns.



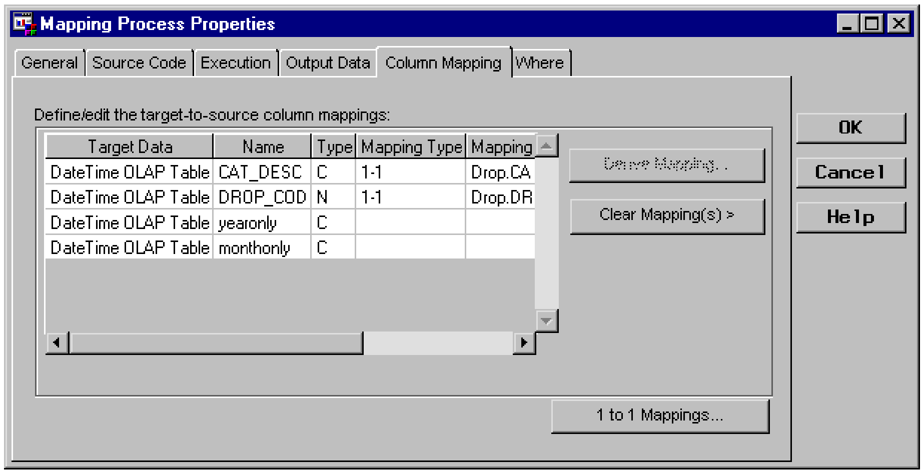
Name	Summary Role	Sort O
CAT_DESC	CLASS	ASCE
DROP_COD	STATISTIC	NONE
yearonly	CLASS	ASCE
monthonly	CLASS	ASCE

```
DateTime OLAP Table
  Class Columns
    CAT_DESC (Sort=ASCENDING
    yearonly (Sort=ASCENDING)
    monthonly (Sort=ASCENDING)
  ID Columns
  Statistic Columns
```

Define Mapping Process in Process Editor Job

Mappings for the imported columns can be defined simply as one-to-one mappings. However, derived mappings must be defined for the columns **yearonly** and **monthonly**. You can easily do this by using stored expressions available in the Expression Builder window from the **DATE/TIME** standard component.

Navigate to the Column Mapping tab on the Mapping Process Properties window as follows:

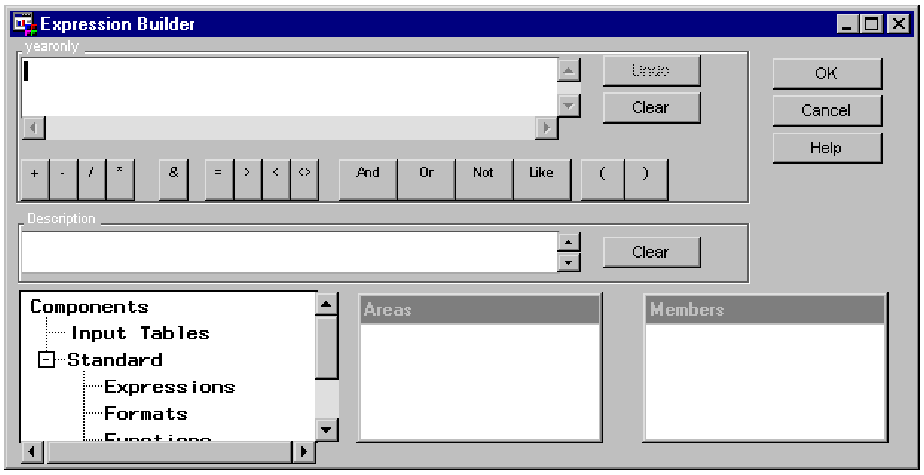


Define/edit the target-to-source column mappings:

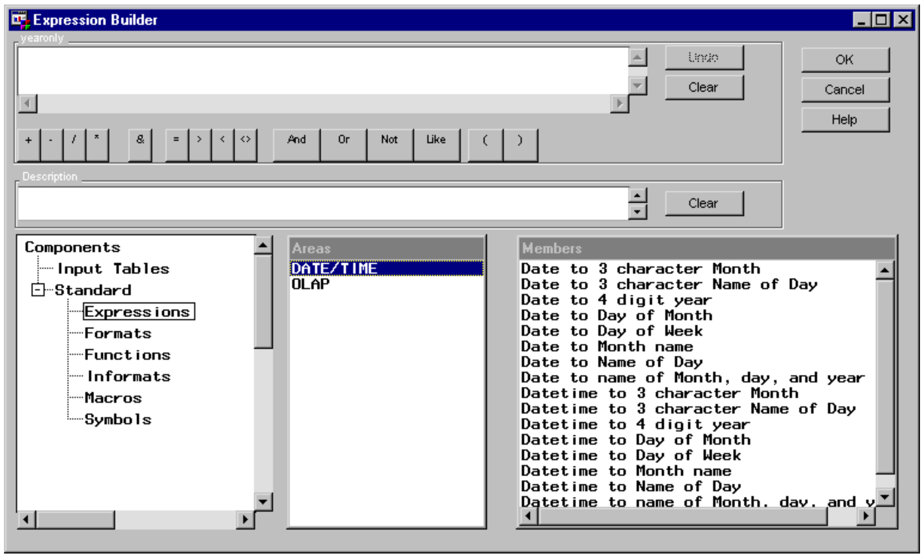
Target Data	Name	Type	Mapping Type	Mapping
DateTime OLAP Table	CAT_DESC	C	1-1	Drop.CA
DateTime OLAP Table	DROP_COD	N	1-1	Drop.DR
DateTime OLAP Table	yearonly	C		
DateTime OLAP Table	monthonly	C		

To define derived mappings:

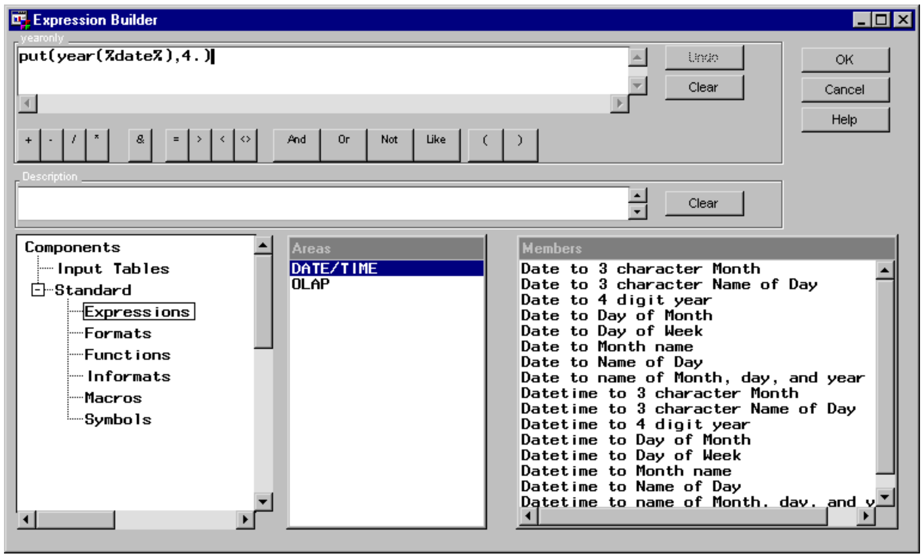
1 Select **`yearonly`**, and then click Derive Mapping, which opens the Expression Builder window.



2 In the Expression Builder window, select the **`Standard Expressions`** component, and then **`DATE/TIME`** to display a list of available stored expressions.

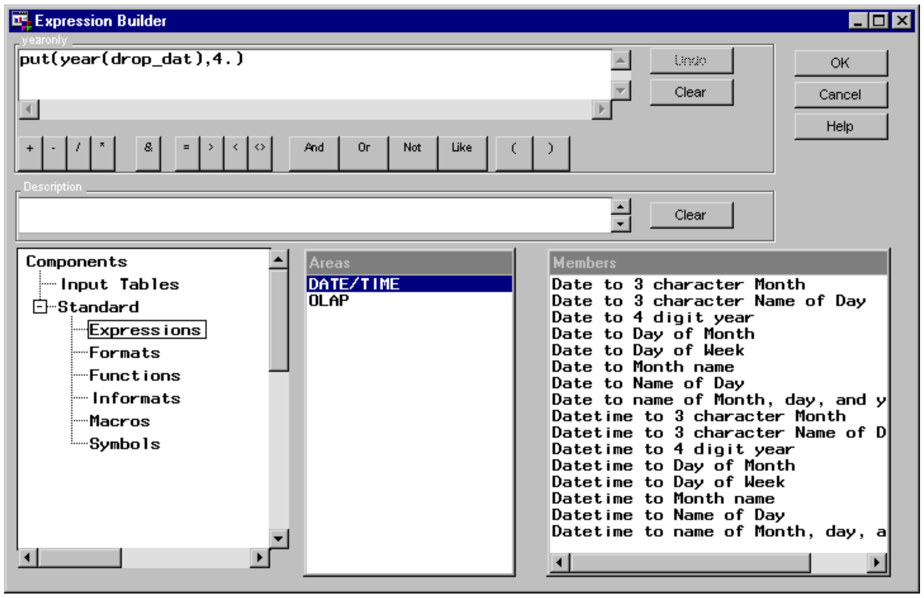


3 Select the stored expression **`Date to 4 digit year`**. The resulting expression uses the PUT and YEAR functions.



```
put(year(%date%),4.)
```

4 You must replace the argument **`%date%`** with the input column name, which is **`drop_dat`**.



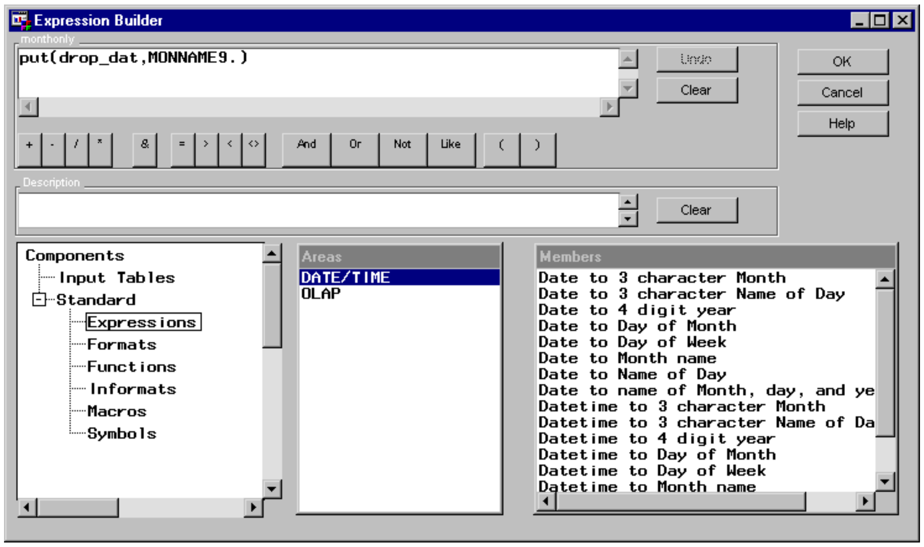
```
put(year(drop_dat),4.)
```

```
Date to 3 character Month
Date to 3 character Name of Day
Date to 4 digit year
Date to Day of Month
Date to Day of Week
Date to Month name
Date to Name of Day
Date to name of Month, day, and y
Datetime to 3 character Month
Datetime to 3 character Name of D
Datetime to 4 digit year
Datetime to Day of Month
Datetime to Day of Week
Datetime to Month name
Datetime to Name of Day
Datetime to name of Month, day, a
```

5 Click OK on the Expression Builder window. The expression is assigned to the column, and you are returned to the Column Mapping tab in the Mapping Process Properties window. Note that SAS/Warehouse Administrator interprets the expression as:

```
yearonly=put(year(drop_dat),4.)
```

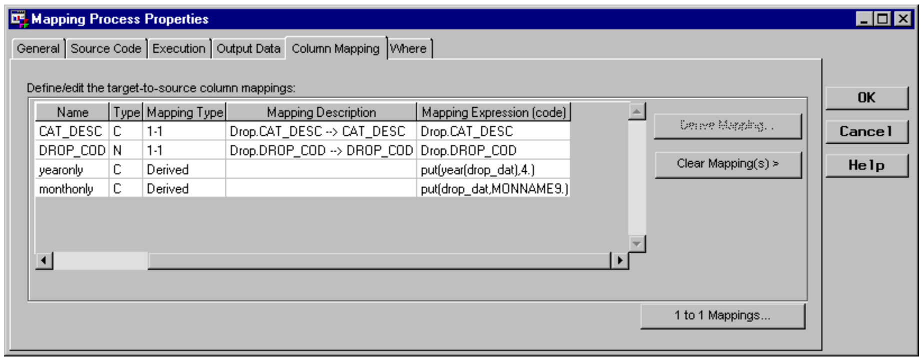
6 Repeat the steps for the **monthonly** column and select the stored expression **Date to Month name**, which results in the following expression:



```
put(drop_dat,MONNAME9.)
```

```
Date to 3 character Month
Date to 3 character Name of Day
Date to 4 digit year
Date to Day of Month
Date to Day of Week
Date to Month name
Date to Name of Day
Date to name of Month, day, and ye
Datetime to 3 character Month
Datetime to 3 character Name of Da
Datetime to 4 digit year
Datetime to Day of Month
Datetime to Day of Week
Datetime to Month name
```

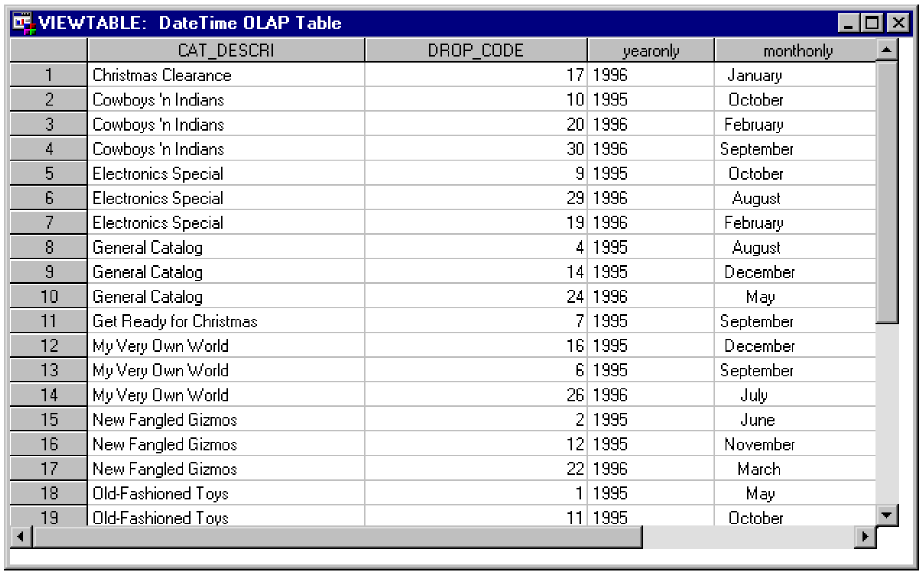
The following Column Mapping tab in the Mapping Process Properties window displays the two columns with derived mapping defined:



Define/edit the target-to-source column mappings:

Name	Type	Mapping Type	Mapping Description	Mapping Expression (code)
CAT_DESC	C	1-1	Drop.CAT_DESC --> CAT_DESC	Drop.CAT_DESC
DROP_COD	N	1-1	Drop.DROP_COD --> DROP_COD	Drop.DROP_COD
yearonly	C	Derived		put(year(drop_dat),4.)
monthonly	C	Derived		put(drop_dat,MONNAME9.)

Once the Job is executed and the table loaded, here is the result:

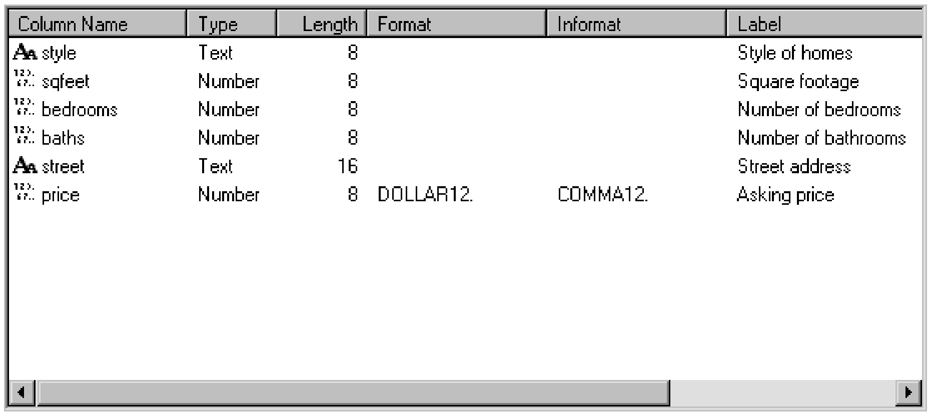


	CAT_DESCRI	DROP_CODE	yearonly	monthonly
1	Christmas Clearance	17	1996	January
2	Cowboys 'n Indians	10	1995	October
3	Cowboys 'n Indians	20	1996	February
4	Cowboys 'n Indians	30	1996	September
5	Electronics Special	9	1995	October
6	Electronics Special	29	1996	August
7	Electronics Special	19	1996	February
8	General Catalog	4	1995	August
9	General Catalog	14	1995	December
10	General Catalog	24	1996	May
11	Get Ready for Christmas	7	1995	September
12	My Very Own World	16	1995	December
13	My Very Own World	6	1995	September
14	My Very Own World	26	1996	July
15	New Fangled Gizmos	2	1995	June
16	New Fangled Gizmos	12	1995	November
17	New Fangled Gizmos	22	1996	March
18	Old-Fashioned Toys	1	1995	May
19	Old-Fashioned Toys	11	1995	October

Example: Using an Input Column for Multiple Summary Roles

Overview

Suppose you want to use a specific column from an input source as both a class column and an analysis column for OLAP summarization. For example, consider an input table that contains the following columns:



Column Name	Type	Length	Format	Informat	Label
style	Text	8			Style of homes
sqfeet	Number	8			Square footage
bedrooms	Number	8			Number of bedrooms
baths	Number	8			Number of bathrooms
street	Text	16			Street address
price	Number	8	DOLLAR12.	COMMA12.	Asking price

You determine that the OLAP data store needs to store two crossings:

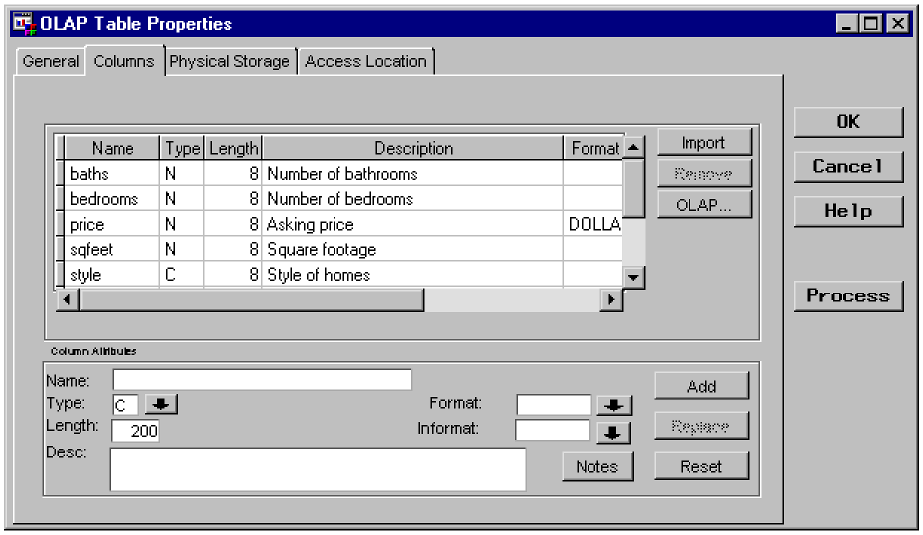
- □ The most detailed level of summarization (an NWAY crossing) uses the class columns: **`style`**, **`bedrooms`**, and **`baths`**. The analysis columns would be **`price`** and **`sqfeet`**.
- □ At a higher level of summarization, you need a crossing based only on the class column **`style`**. The analysis columns would be **`price`**, **`sqfeet`**, **`bedrooms`**, and **`baths`**.

Therefore, the input columns **`bedrooms`** and **`baths`** are used as both class columns and analysis columns. To produce the appropriate results:

1. Import the columns from the input table to the OLAP data store and add two columns: **`bedsmin`** and **`bathsmin`**.
2. Assign OLAP summary roles to the imported columns and to the added columns. That is, the imported columns **`baths`** and **`bedrooms`** are class columns and the added columns **`bedsmin`** and **`bathsmin`** are statistic columns.
3. Assign the desired statistics to the statistic columns. For example, assign the MIN statistic to the **`bedsmin`** and **`bathsmin`** columns.
4. Map the input columns to the appropriate output columns. For example, the input column **`baths`** is mapped to the output columns **`baths`**, which is a class column, and **`bathsmin`**, which is a statistic column.

Import and Add Columns to OLAP Summary Data Store

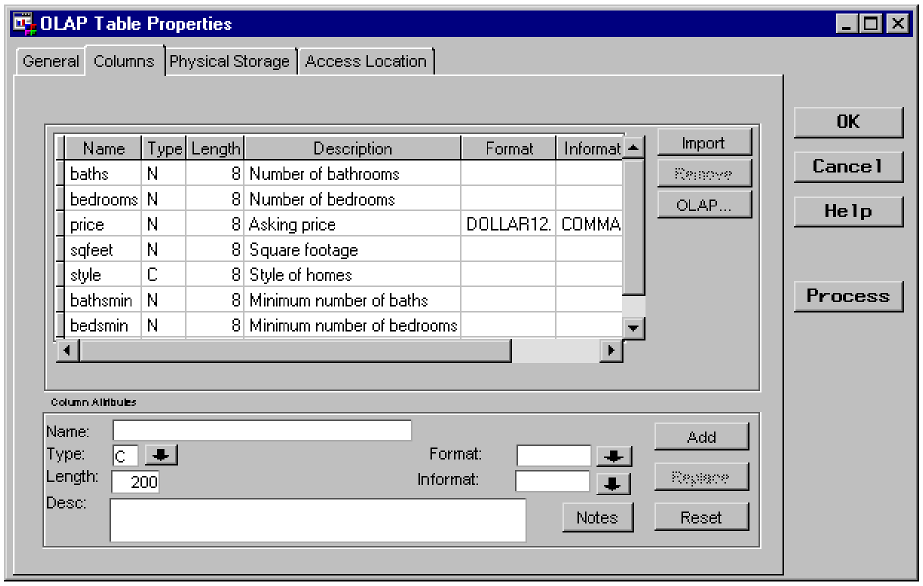
The Columns tab in the object's properties window specifies the columns to be included in an OLAP summary data store, which for this example is the OLAP Table Properties window. This example assumes that the appropriate columns from the input source have been imported as follows:



Name	Type	Length	Description	Format
baths	N	8	Number of bathrooms	
bedrooms	N	8	Number of bedrooms	
price	N	8	Asking price	DOLLA
sqfeet	N	8	Square footage	
style	C	8	Style of homes	

Name:
Type: C
Length: 200
Desc:
Format:
Informat:

To add a column that does not exist in the input data, define the column's attributes, and then click Add. (Note that you can also select an existing column, change its name, and then click Add to easily copy a column.) The result of adding columns **bathsmin** and **bedsmin** is as follows:

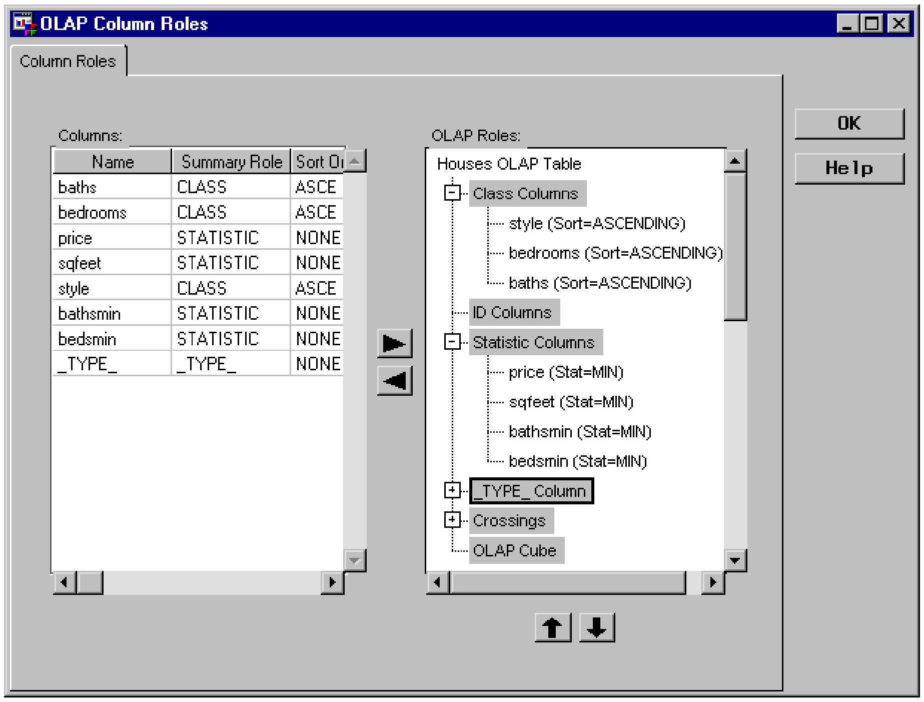


Name	Type	Length	Description	Format	Informat
baths	N	8	Number of bathrooms		
bedrooms	N	8	Number of bedrooms		
price	N	8	Asking price	DOLLAR12.	COMMA
sqfeet	N	8	Square footage		
style	C	8	Style of homes		
bathsmin	N	8	Minimum number of baths		
bedsmin	N	8	Minimum number of bedrooms		

Name:
Type: C
Length: 200
Desc:
Format:
Informat:

Assign OLAP Summary Roles to the Columns

On the Columns tab, click OLAP to open the OLAP Column Roles window from which you assign OLAP summary roles. The assigned summary roles are as follows:


OLAP Column Roles

Column Roles

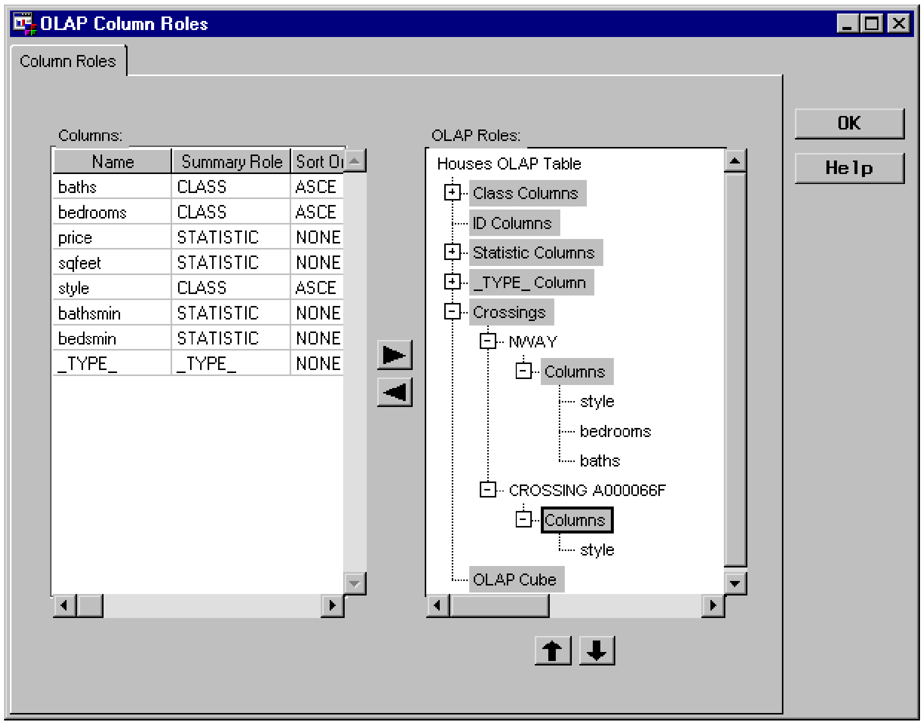
Columns:

Name	Summary Role	Sort O
baths	CLASS	ASCE
bedrooms	CLASS	ASCE
price	STATISTIC	NONE
sqfeet	STATISTIC	NONE
style	CLASS	ASCE
bathsmin	STATISTIC	NONE
bedsmin	STATISTIC	NONE
TYPE	_TYPE_	NONE

OLAP Roles:

```
Houses OLAP Table
  Class Columns
    style (Sort=ASCENDING)
    bedrooms (Sort=ASCENDING)
    baths (Sort=ASCENDING)
  ID Columns
  Statistic Columns
    price (Stat=MIN)
    sqfeet (Stat=MIN)
    bathsmin (Stat=MIN)
    bedsmin (Stat=MIN)
  _TYPE_ Column
  Crossings
  OLAP Cube
```

The following screen displays the defined crossings for the data store:


OLAP Column Roles

Column Roles

Columns:

Name	Summary Role	Sort O
baths	CLASS	ASCE
bedrooms	CLASS	ASCE
price	STATISTIC	NONE
sqfeet	STATISTIC	NONE
style	CLASS	ASCE
bathsmin	STATISTIC	NONE
bedsmin	STATISTIC	NONE
TYPE	_TYPE_	NONE

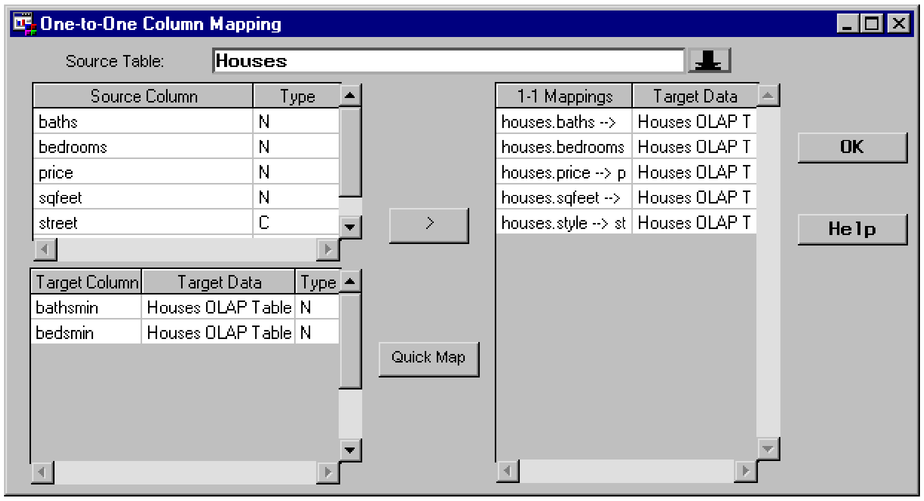
OLAP Roles:

```
Houses OLAP Table
  Class Columns
  ID Columns
  Statistic Columns
  _TYPE_ Column
  Crossings
    NWAY
      Columns
        style
        bedrooms
        baths
    CROSSING A000066F
      Columns
        style
  OLAP Cube
```

Define Mapping Process in Process Editor Job

Column mapping is defined as one-to-one mapping. To map the columns:

1 From the Column Mapping tab in the Mapping Process Properties window, click 1 to 1 Mappings, which opens the One-to-One Column Mapping window, and then click Quick Map, which produces the following results. Note that the two added columns **bathsmin** and **bedsmin** are not mapped.



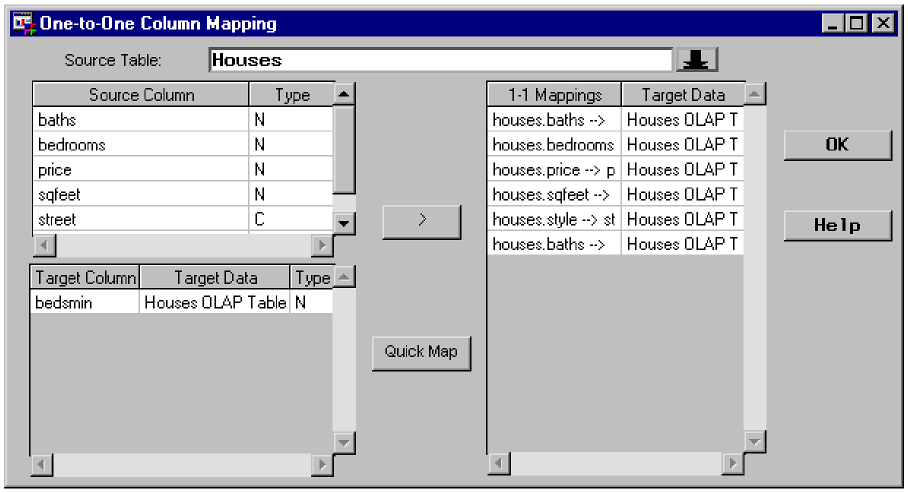
Source Table: Houses

Source Column	Type
baths	N
bedrooms	N
price	N
sqfeet	N
street	C

Target Column	Target Data	Type
bathsmin	Houses OLAP Table	N
bedsmin	Houses OLAP Table	N

1-1 Mappings	Target Data
houses.baths -->	Houses OLAP T
houses.bedrooms	Houses OLAP T
houses.price --> p	Houses OLAP T
houses.sqfeet -->	Houses OLAP T
houses.style --> st	Houses OLAP T

2 Select the input column **baths** from **Source Column**, select the output column **bathsmin** from **Target Column**, and then click the right arrow to map the two columns.



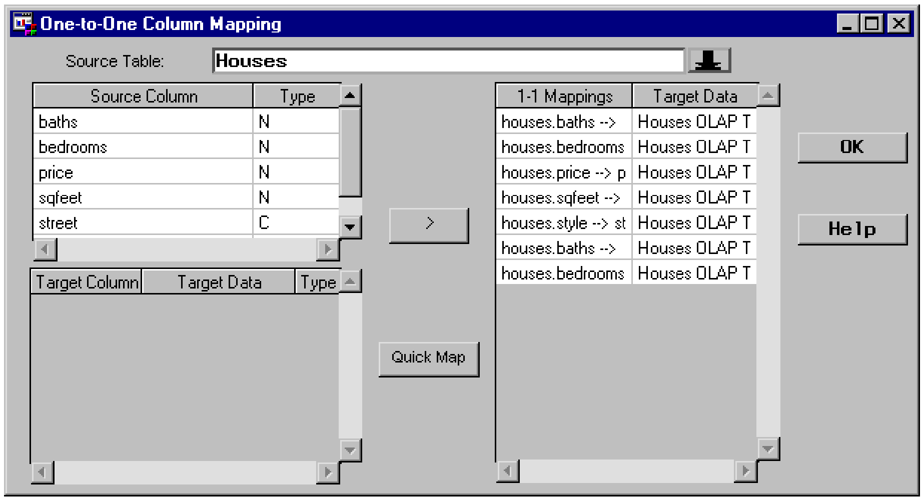
Source Table: Houses

Source Column	Type
baths	N
bedrooms	N
price	N
sqfeet	N
street	C

Target Column	Target Data	Type
bedsmin	Houses OLAP Table	N

1-1 Mappings	Target Data
houses.baths -->	Houses OLAP T
houses.bedrooms	Houses OLAP T
houses.price --> p	Houses OLAP T
houses.sqfeet -->	Houses OLAP T
houses.style --> st	Houses OLAP T
houses.baths -->	Houses OLAP T

3 Select the input column **bedrooms** from **Source Column**, select the output column **bedsmin** from **Target Column**, and then click the right arrow to map the two columns.



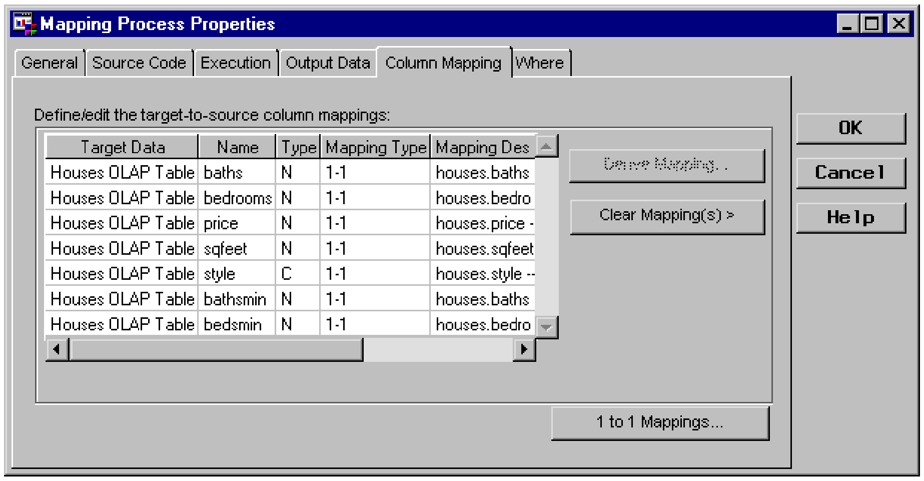
Source Table: Houses

Source Column	Type
baths	N
bedrooms	N
price	N
sqfeet	N
street	C

Target Column	Target Data	Type

1-1 Mappings	Target Data
houses.baths -->	Houses OLAP T
houses.bedrooms	Houses OLAP T
houses.price --> p	Houses OLAP T
houses.sqfeet -->	Houses OLAP T
houses.style --> st	Houses OLAP T
houses.baths -->	Houses OLAP T
houses.bedrooms	Houses OLAP T

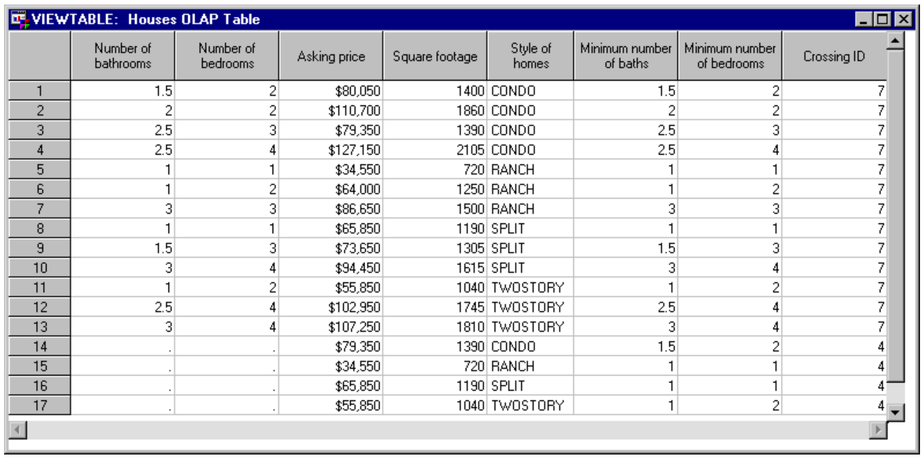
When you click OK, you are returned to the Column Mapping tab in the Mapping Process Properties window as follows:



Define/edit the target-to-source column mappings:

Target Data	Name	Type	Mapping Type	Mapping Des
Houses OLAP Table	baths	N	1-1	houses.baths
Houses OLAP Table	bedrooms	N	1-1	houses.bedro
Houses OLAP Table	price	N	1-1	houses.price -
Houses OLAP Table	sqfeet	N	1-1	houses.sqfeet
Houses OLAP Table	style	C	1-1	houses.style --
Houses OLAP Table	bathsmin	N	1-1	houses.baths
Houses OLAP Table	bedsmin	N	1-1	houses.bedro

After the Job is executed and the table is loaded, the results are displayed as follows:



	Number of bathrooms	Number of bedrooms	Asking price	Square footage	Style of homes	Minimum number of baths	Minimum number of bedrooms	Crossing ID
1	1.5	2	$80,050	1400	CONDO	1.5	2	7
2	2	2	$110,700	1860	CONDO	2	2	7
3	2.5	3	$79,350	1390	CONDO	2.5	3	7
4	2.5	4	$127,150	2105	CONDO	2.5	4	7
5	1	1	$34,550	720	RANCH	1	1	7
6	1	2	$64,000	1250	RANCH	1	2	7
7	3	3	$86,650	1500	RANCH	3	3	7
8	1	1	$65,850	1190	SPLIT	1	1	7
9	1.5	3	$73,650	1305	SPLIT	1.5	3	7
10	3	4	$94,450	1615	SPLIT	3	4	7
11	1	2	$55,850	1040	TWOSTORY	1	2	7
12	2.5	4	$102,950	1745	TWOSTORY	2.5	4	7
13	3	4	$107,250	1810	TWOSTORY	3	4	7
14	.	.	$79,350	1390	CONDO	1.5	2	4
15	.	.	$34,550	720	RANCH	1	1	4
16	.	.	$65,850	1190	SPLIT	1	1	4
17	.	.	$55,850	1040	TWOSTORY	1	2	4

What’s Next

After you create summary data stores, you can exploit them with tools designed to work with summary data, such as SAS/EIS software. See “Example: Exporting Metadata to SAS/EIS Software” on page 316.

For OLAP Tables only, you can use them as inputs to other SAS/Warehouse Administrator objects such as Data Tables and other summary data stores.

CHAPTER 12 Maintaining Information Marts

Overview

SAS/Warehouse Administrator provides the ability to create an *information mart*, which is an application (or the output from an application) that runs against warehouse data. An information mart can provide answers to business questions and can also be used to document any portion of the warehouse or its implementation.

In SAS/Warehouse Administrator, the following objects are provided to create an information mart:

Information Mart (or InfoMart)	is a simple grouping element that allows you to organize Information Mart Items and Information Mart Files. Unlike most objects in SAS/Warehouse Administrator, Information Marts are used to display information rather than store it. You can add an Information Mart to a Subject, a Data Group, or an ODD Group.
Information Mart Item (or InfoMart Item)	is a metadata record that specifies a routine, which generates output from data stores in a Data Warehouse. The output is usually a SAS chart, report, graph, or query result. For example, an Information Mart Item could display a chart that summarizes sales information from a detail data store. An Information Mart Item can be added only to an Information Mart.
Information Mart File (or InfoMart File)	is a metadata record that specifies a file type other than SAS that is registered in a Warehouse Environment. The file can be a spreadsheet, an HTML report, or any file that can be opened by an external application. Information Mart File metadata describes the

location of an external file and the technique for opening that file. For example, an Information Mart File could open a spreadsheet that contains information useful to the person managing a Warehouse Environment. An Information Mart File can be added only to an Information Mart.

In general, to maintain Information Marts:

1 Define the properties of the Information Mart.
2 Define the properties of Information Mart Item(s) and Information Mart File(s).
3 For an Information Mart Item, define a Process Editor Job that includes the data preparation processes, which prepare the data to be loaded into the data store(s), and the Load Step, which defines the steps to load the data store(s).
4 For an Information Mart Item, execute the Job.
5 Verify that the data store is loaded; that is, check logs, use the data utilities, and so on.

Note: The basic steps for creating an Information Mart and related objects are described in the online Help. To display the relevant online Help, in the SAS System Help contents, select **`Help on SAS Software Products`**, then select **`Using SAS/Warehouse Administrator Software`**. Finally, select **`Defining Information Marts`**. In addition, you can display Help for most SAS/Warehouse Administrator windows by clicking Help on the window. △

Preparing to Create Information Marts

Before you create an Information Mart and its associated Information Mart Items and Information Mart Files, you must do some preparation, which is summarized as follows:

Hierarchy of objects	In SAS/Warehouse Administrator, make sure that you have created the appropriate Data Warehouse Environment, Data Warehouse, and Subject, Data Group, or ODD Group.
Input source(s)	Determine which input source(s) to use for each Information Mart Item and Information Mart File. For example, if the input source for an Information Mart File is an ODD, the ODD must be fully operational.
Processes	For an Information Mart Item, determine the user-written source code, which produces the output, and determine the SAS catalog entry in which to store it. Determine whether you need to define processes such as a User Exit process, Data Transfer process, or Record selector process. Information Marts do not have Mapping processes.
Output target	Determine the appropriate location for each Information Mart Item and Information Mart File. Determine on what platform you want to store the Information Mart File and Information Mart Item. That is, you could store the data store locally or on a remote host.

Example: Creating an Information Mart

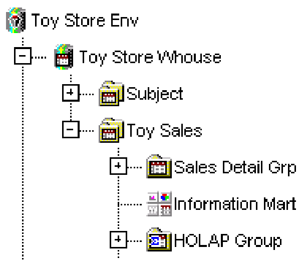
Overview

This example creates an Information Mart to group related Information Mart Items and Information Mart Files.

Note: The following explanations describe the metadata and methods used to achieve the desired results. It is assumed that the appropriate Data Warehouse Environment, Data Warehouse, and Subject already exist. △

Define Information Mart Properties

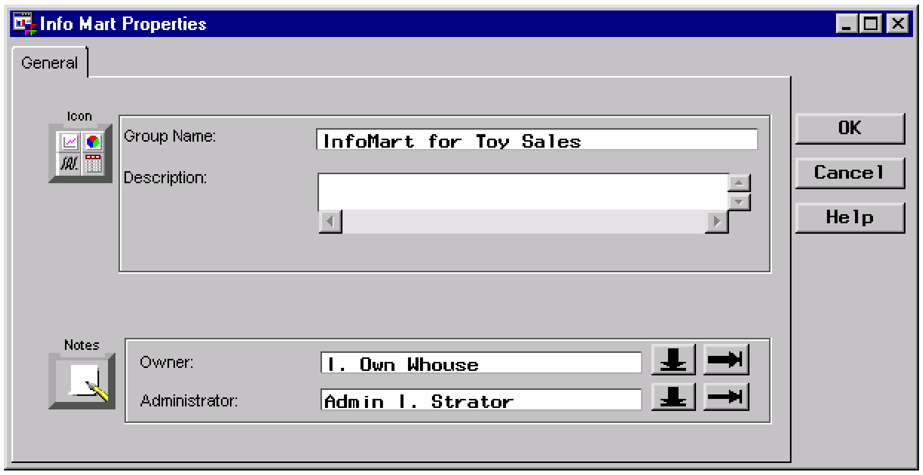
In the SAS/Warehouse Administrator Explorer, position the cursor on the Subject (for example, **`Toy Sales`**), click the right mouse button, select **`Add Item`**, and then **`Information Mart`**. In the Explorer window, a new Information Mart is added under the Subject as follows:



```
Toy Store Env
  Toy Store Whouse
    Subject
    Toy Sales
      Sales Detail Grp
      Information Mart
      HOLAP Group
```

To update the default metadata for the Information Mart, position the cursor on its icon, click the right mouse button, and select **`Properties`**. The Info Mart Properties window displays for you to enter the appropriate information.

General Tab	specifies the object name **`InfoMart for Toy Sales`**, a description, an owner, and an administrator.



Info Mart Properties

General

Group Name: InfoMart for Toy Sales

Description:

Owner: I. Own Whouse

Administrator: Admin I. Strator

The next sections describe how to create an Information Mart Item and an Information Mart File.

Example: Creating an Information Mart Item

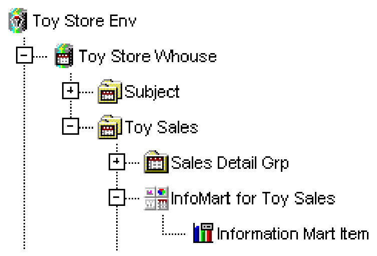
Overview

This example creates an Information Mart Item that displays a percentage chart of sale promotion types generated from the ODD **Promotions**.

Note: The following explanations describe the metadata and methods used to achieve the desired results. It is assumed that the appropriate Data Warehouse Environment, Data Warehouse, Subject, Information Mart, and ODD already exist. △

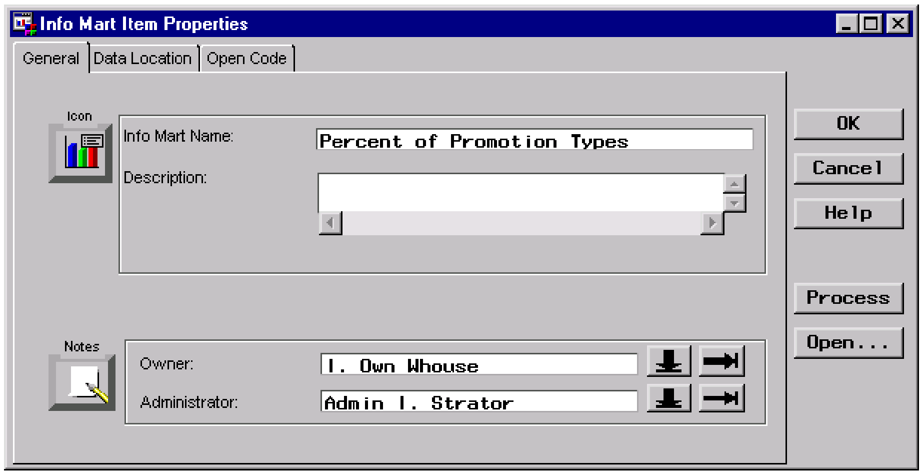
Define Information Mart Item Properties

In the SAS/Warehouse Administrator Explorer, position the cursor on the Information Mart (for example, **InfoMart for Toy Sales**), click the right mouse button, select **Add Item**, and then **Information Mart Item**. In the Explorer window, a new Information Mart Item is added under the Information Mart as follows:



To update the default metadata for the item, position the cursor on its icon, click the right mouse button, and select **Properties**. The Info Mart Item Properties window displays for you to enter the appropriate information.

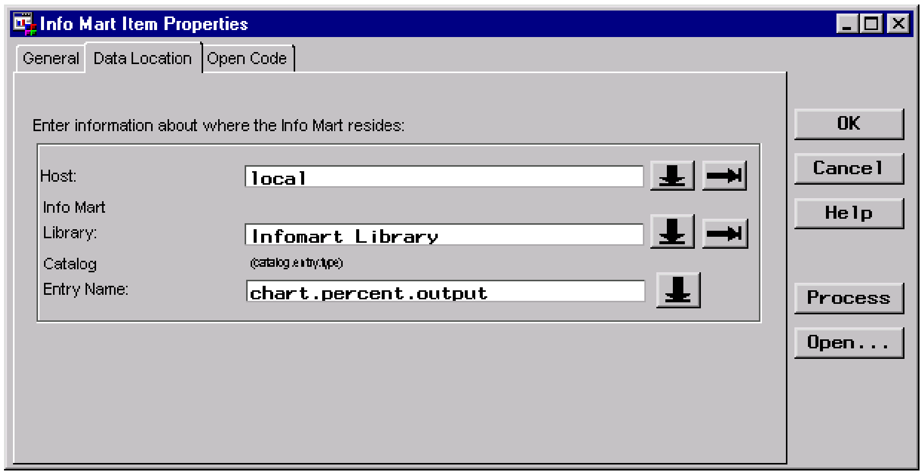
General Tab specifies the item name **Percent of Promotion Types**, a description, an owner, and an administrator.



Info Mart Name: Percent of Promotion Types
Description:
Owner: I. Own Whouse
Administrator: Admin I. Strator

Data Location Tab specifies where the output is stored. An Information Mart Item is stored as a SAS catalog entry. The following types are supported:

SOURCE, FRAME, PROGRAM, GRSEG, LOG, OUTPUT, QUERY, and REPORT.



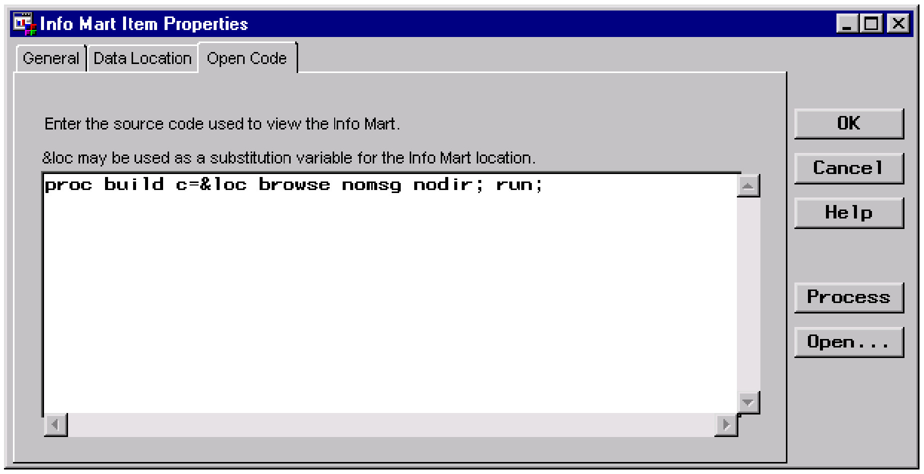
Enter information about where the Info Mart resides:

Host: local

Info Mart Library: Infomart Library

Catalog Entry Name: chart.percent.output (catalog.entry.type)

Open Code Tab specifies the source code used to view the item stored in the SAS catalog entry listed in the Data Location tab.



Enter the source code used to view the Info Mart.

&loc may be used as a substitution variable for the Info Mart location.

```
proc build c=&loc browse nomsg nodir; run;
```

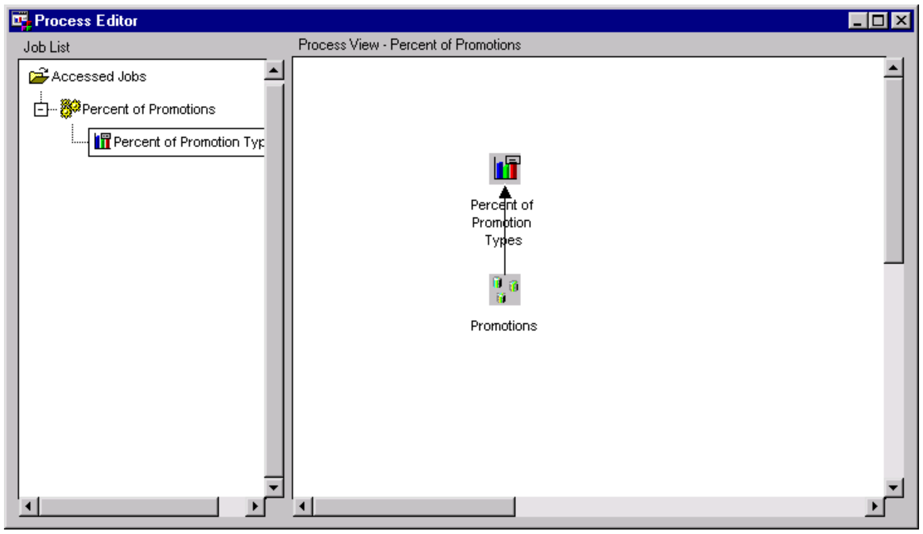
Note: To use the default code shown in this screen, you must have SAS/AF software licensed. Other examples of code that you could specify include the WBROWSE command or, for SOURCE catalog entries, you could specify

```
dm ''notepad &loc'';
```

△

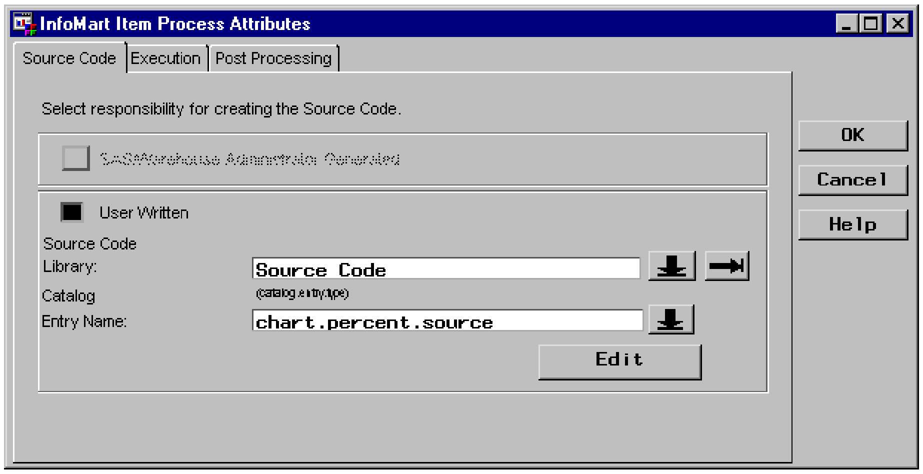
Define Process Editor Job

In the Process Editor Job, the Information Mart Item **`Percent of Promotion Types`** is specified as the output target and the ODD **`Promotions`** is specified as the input source. The following Process Editor window displays the Process Flow for the Job:



The Load process defined in the Job is summarized next. Note that there is not a Mapping process. The Information Mart Item is generated by running source code against the specified data.

Load Step Process	The source code is user written and stored as a SAS catalog entry.



Select responsibility for creating the Source Code.

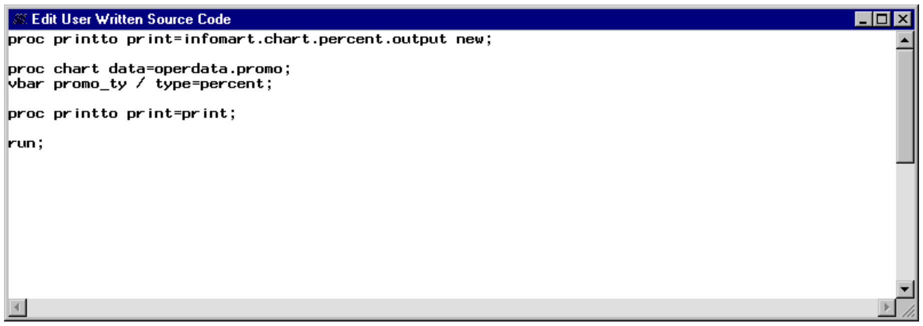
☐ SAS/Warehouse Administrator Generated

■ User Written

Source Code Library: Source Code

Catalog Entry Name: chart.percent.source (catalog.entry.type)

To enter the source code or to display it, click Edit. The user-written source code is as follows:

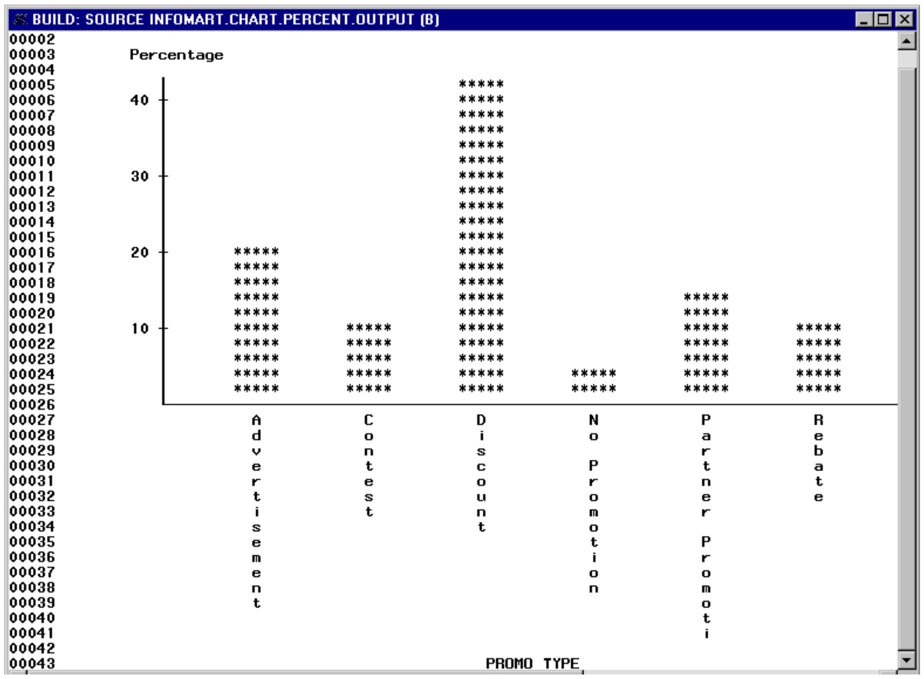


```
proc printto print=infomart.chart.percent.output new;

proc chart data=operdata.promo;
vbar promo_ty / type=percent;

proc printto print=print;

run;
```

For more information about Process Editor Jobs, see Chapter 13, "Maintaining Jobs," on page 251. For more information about processes, see Chapter 14, "Maintaining Processes," on page 281.

Display Output

First, you must execute the Job, which executes the user-written code specified in the Load process, generates the output, and stores it in the specified catalog entry. Then, to display the output, open the Info Mart Item Properties window and click Open.



```
       Percentage

          |                               *****
       40 +                               *****
          |                               *****
          |                               *****
          |                               *****
          |                               *****
       30 +                               *****
          |                               *****
          |                               *****
          |                               *****
          |                               *****
       20 +       *****                   *****
          |       *****                   *****
          |       *****                   *****
          |       *****                   *****                   *****
          |       *****                   *****                   *****
       10 +       *****       *****       *****                   *****       *****
          |       *****       *****       *****                   *****       *****
          |       *****       *****       *****                   *****       *****
          |       *****       *****       *****       *****       *****       *****
          |       *****       *****       *****       *****       *****       *****
          +------------------------------------------------------------------------------
                    A           C           D           N           P           R
                    d           o           i           o           a           e
                    v           n           s                       r           b
                    e           t           c           P           t           a
                    r           e           o           r           n           t
                    t           s           u           o           e           e
                    i           t           n           m           r
                    s                       t           o
                    e                                   t           P
                    m                                   i           r
                    e                                   o           o
                    n                                   n           m
                    t                                               o
                                                                    t
                                                                    i

                                             PROMO TYPE
```

Example: Creating an Information Mart File

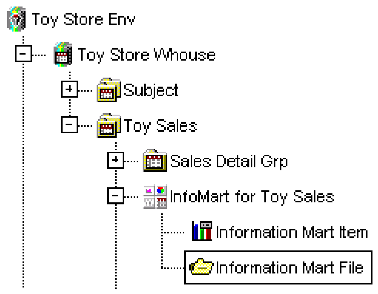
Overview

This example creates an Information Mart File that accesses a document.

Note: The following explanations describe the metadata and methods used to achieve the desired results. It is assumed that the appropriate Data Warehouse Environment, Data Warehouse, Subject, Information Mart, and document already exist. △

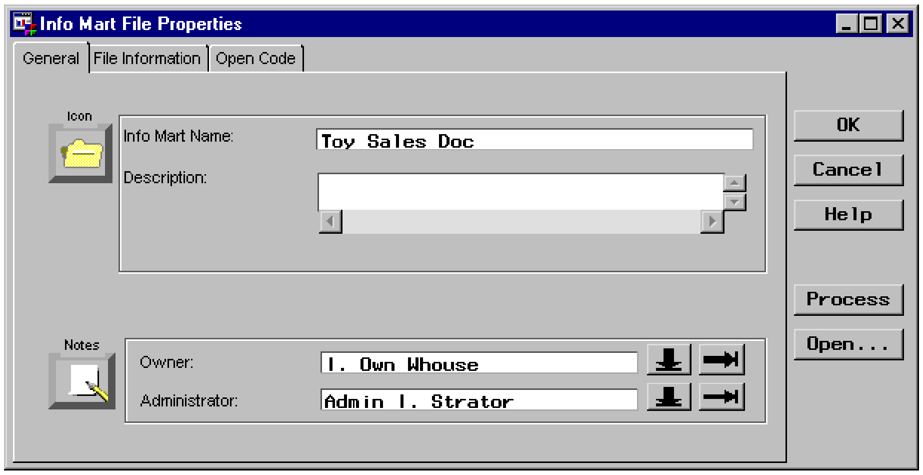
Define Information Mart File Properties

In the SAS/Warehouse Administrator Explorer, position the cursor on the Information Mart (for example, **`InfoMart for Toy Sales`**), click the right mouse button, select **`Add Item`**, and then **`Information Mart File`**. In the Explorer window, a new Information Mart File is added under the Information Mart as follows:



To update the default metadata for the file, position the cursor on its icon, click the right mouse button, and select **`Properties`**. The Info Mart File Properties window displays for you to enter the appropriate information.

General Tab — specifies the file name **`Toy Sales Doc`**, a description, an owner, and an administrator.



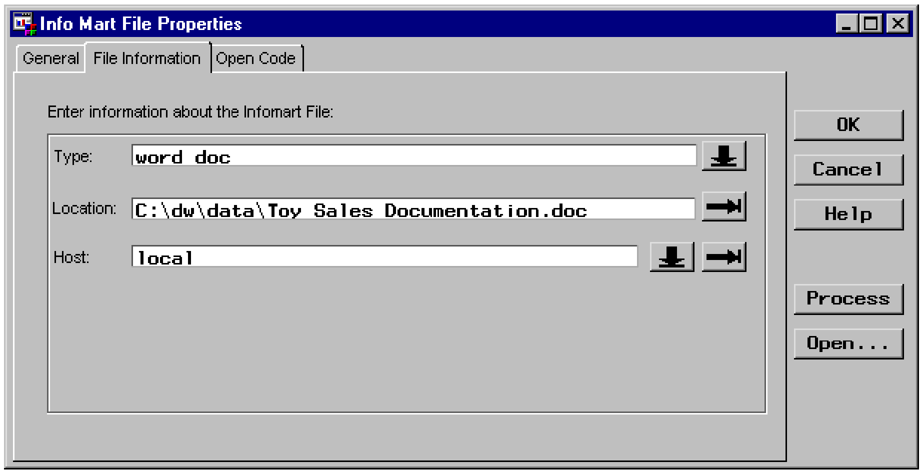
Info Mart Name: Toy Sales Doc

Description:

Owner: I. Own Whouse

Administrator: Admin I. Strator

File Information Tab — specifies where the file is stored and what type of file it is.



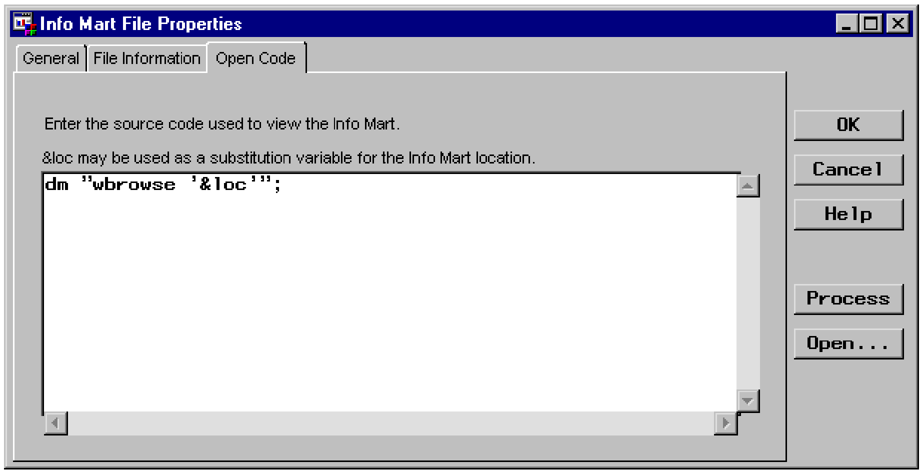
Enter information about the Infomart File:

Type: word doc

Location: C:\dw\data\Toy Sales Documentation.doc

Host: local

Open Code Tab — specifies the source code used to view the file.



Enter the source code used to view the Info Mart.

&loc may be used as a substitution variable for the Info Mart location.

```
dm "wbrowse '&loc'";
```

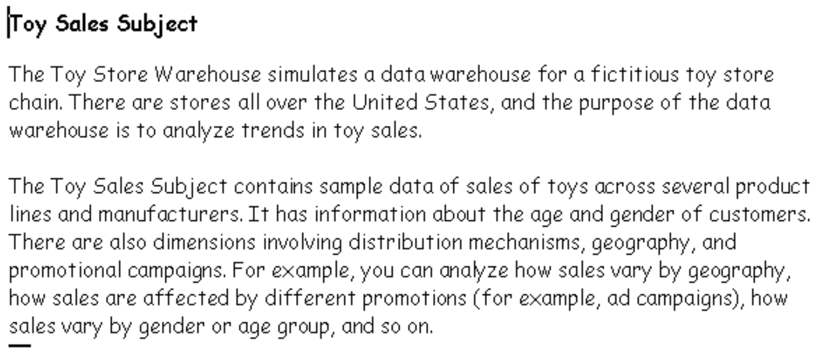
Note: Another example of code that you could use to specify Microsoft Word documents is:

```
dm ''x winword.exe &loc'';
```

△

Display Output

To display the contents of an Information Mart File, open the Info Mart File Properties window and click Open.



Toy Sales Subject

The Toy Store Warehouse simulates a data warehouse for a fictitious toy store chain. There are stores all over the United States, and the purpose of the data warehouse is to analyze trends in toy sales.

The Toy Sales Subject contains sample data of sales of toys across several product lines and manufacturers. It has information about the age and gender of customers. There are also dimensions involving distribution mechanisms, geography, and promotional campaigns. For example, you can analyze how sales vary by geography, how sales are affected by different promotions (for example, ad campaigns), how sales vary by gender or age group, and so on.

PART 4

General Tasks

CHAPTER
13

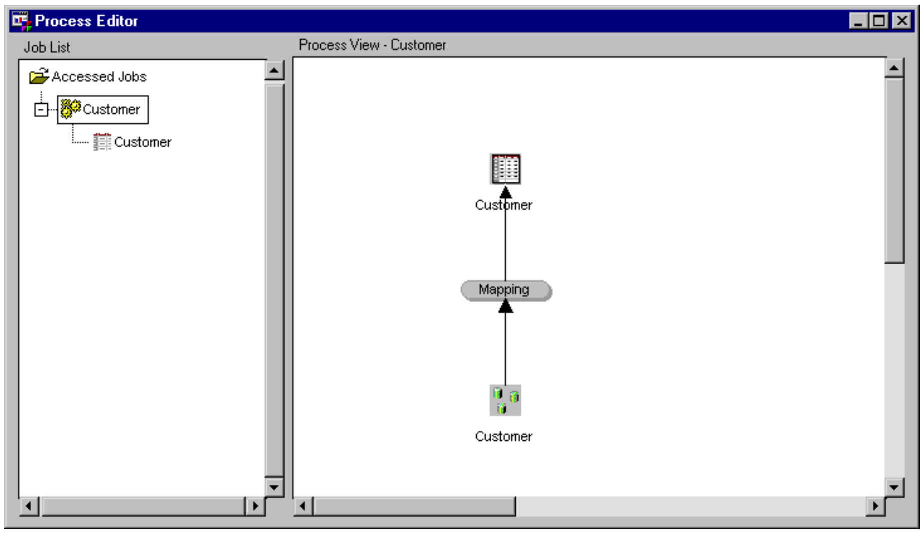
Maintaining Jobs

Overview

This chapter assumes that you are familiar with the basic information about Jobs as described in “Jobs” on page 50 and are ready to define the Jobs to create your data stores. A Job enables you to connect a series of process steps into a single unit. You can specify the processes for a Job in two ways:

- Define a Process Flow in the Process Editor. If a Process Flow exists, SAS/Warehouse Administrator will generate code for the Job.

  A Process Flow is a diagram composed of symbols with connecting arrows and descriptive text that illustrate how data moves from input source(s) to output table(s) and what extractions and transformations occur in between. The icons represent the data stores, and the text boxes represent data preparation processes. The top icon is the output table that is created by the Job. In the Process Flow, data moves from the bottom to the top as shown in the following Process Flow:



- Supply user-written code that contains the processes to create the data store(s). When you supply the code, a Process Flow is not required. However, you might want to define one for documentation purposes and to take advantage of the ability of SAS/Warehouse Administrator in that it can generate LIBNAME statements for the input sources specified in a Process Flow. For an example, see “Define Process Editor Job” on page 124, which illustrates defining a Process Flow for creating an ODD in which user-written code is supplied.

In addition to specifying processes, a Job may also include the following:

- scheduling metadata, which enables the Job to be executed in batch mode at a specified date and time. See the “Scheduling Jobs” chapter in this document.
- a Job Flow, which is a user-defined diagram in the Job View of the Process Editor that defines metadata for Job dependencies. It is composed of symbols, with connecting arrows and descriptive text, that illustrate the sequence in which Jobs and Events are executed. Job Flows are not required.

  Note: The current release of SAS/Warehouse Administrator does not generate code for Job Flows. To use them, you must write a Metadata API program that reads Job Flows and generates code for them. For details, see *SAS/Warehouse Administrator Metadata API Reference, Release 2.3*. △

In general, to maintain Jobs, do the following:

1. Add a Job for the data store.
2. Define the Job properties, such as whether the Job code will be generated by SAS/Warehouse Administrator or supplied by user-written code.
3. For SAS/Warehouse Administrator to generate code, define the Job Process Flow, which initially requires the output table and an input source. Typically, a Mapping process is added automatically to the Process Flow when you specify an input source. You must then define the Mapping properties.
4. For a Process Flow, you might need to add additional output tables and input sources, which also require you to update the Mapping properties.
5. For a Process Flow, after you have output table(s) and input source(s) specified, you might need to specify additional processes to further prepare data for loading into the output table(s). These include User Exit, Data Transfer, and Record Selector processes, which require you to define their properties.
6. For a Process Flow, if necessary, you might need to edit the Load process properties. Note that the code to perform the Load process can be generated by SAS/Warehouse Administrator or supplied by user-written code, except for an ODD or a Detail Logical Table, which must be user-written.
7. Execute the Job interactively, or schedule the Job to run in batch mode at a future date and time.

This chapter focuses on the general procedures for maintaining Jobs. For unique information about Jobs for a specific data store, see the examples in the chapter associated with the data store.

Note: The basic steps for defining a Job are described in the online Help for each type of data store. To display the relevant online Help, in the SAS System Help contents, select

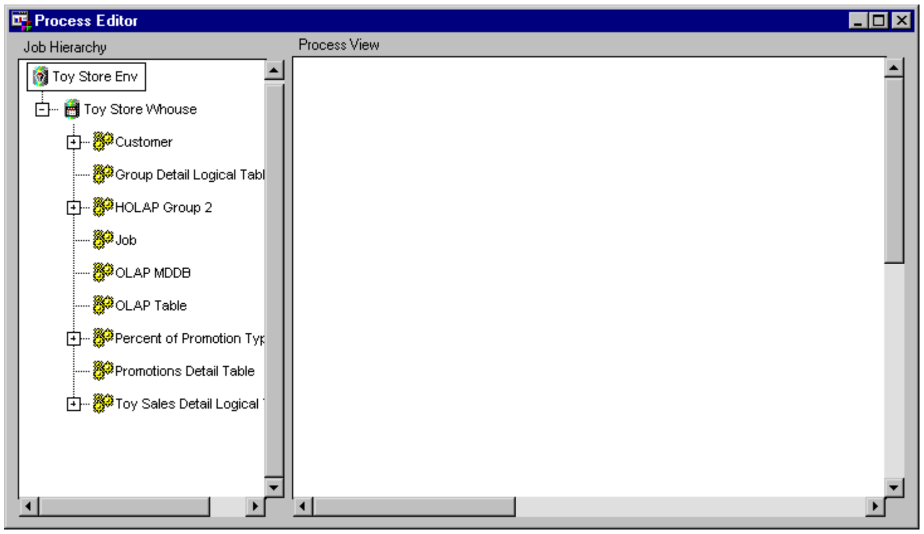
Help on SAS Software Products ► Using SAS/Warehouse Administrator Software

Finally, select defining a specific type of data store, and then select the topic for adding and updating the data store Job. In addition, you can display help for most SAS/Warehouse Administrator windows by clicking Help on the window. △

Using the Process Editor Window

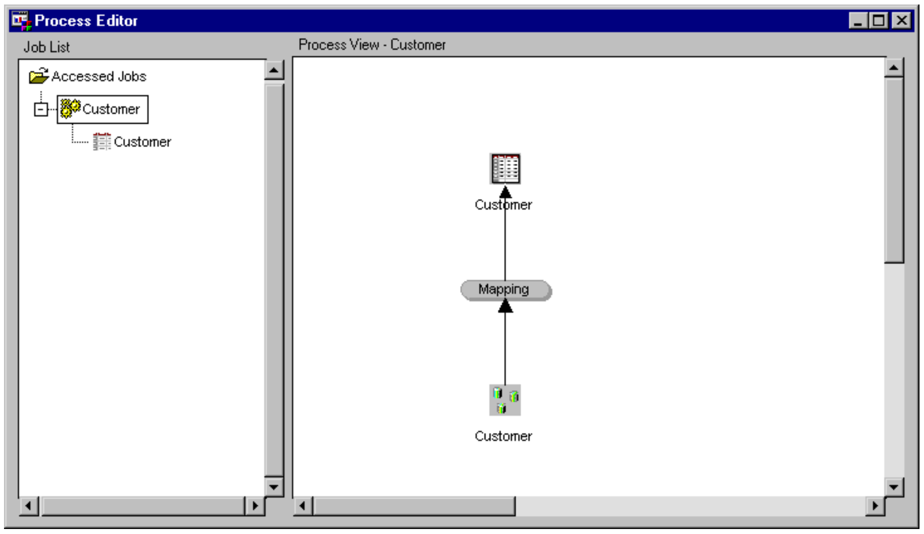
Use the Process Editor window to manage Jobs. The Process Editor window contains a left panel and a right panel, and each panel contains two views as follows:

Left Panel: Job Hierarchy	displays all of the Jobs that are defined in the current Warehouse Environment. To display this view, from the SAS/Warehouse Administrator menu bar, select Tools ► Process Editor



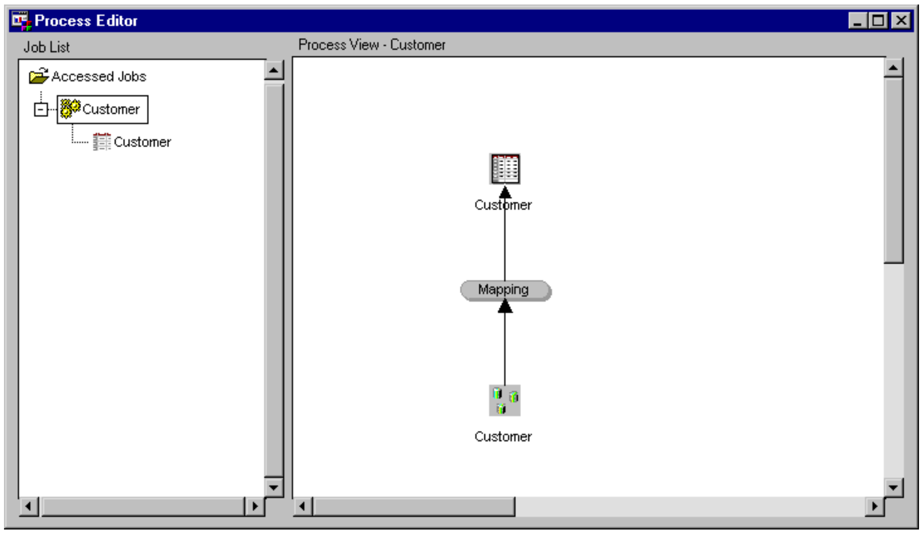
To change from Job Hierarchy to Job List, first close the Process Editor. Then, in the Explorer, click the desired data store with the right mouse button and select **Process**. The Job List for that data store will display.

Left Panel: Job List
displays the Job that creates a specific data store. To display this view, in the Explorer, click the desired data store with the right mouse button and select **Process**.



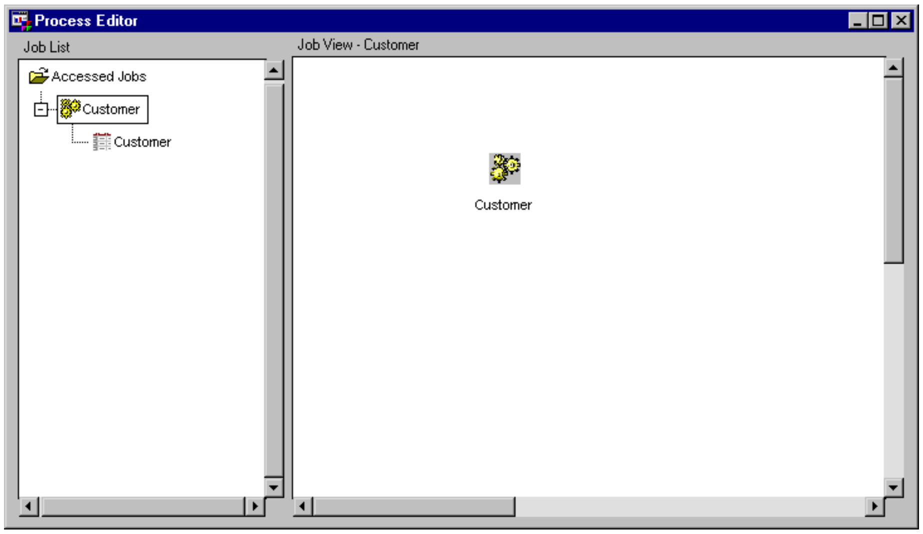
To change from Job List to Job Hierarchy, right-click in the background, select **View**, and then **Job Hierarchy**.

Right Panel: Process View
displays the Process Flow for the Job selected in the Job List or Job Hierarchy. If no Process Flow exists, the Process View is blank. If an icon in a Process Flow has no inputs, it might represent a data store that is created by another Job.



To change from Process View to Job View, click the right mouse button in background and select **`Job View`**.

Right panel: Job View — displays the Job Flow for the Job selected in the Job List or Job Hierarchy. If no Job Flow has been defined for the Job, the Job View displays the active Job.



To change from Job View to Process View, click the right mouse button in background and select **`Process View`**.

Example: Defining a Job for which SAS/Warehouse Administrator Generates Code

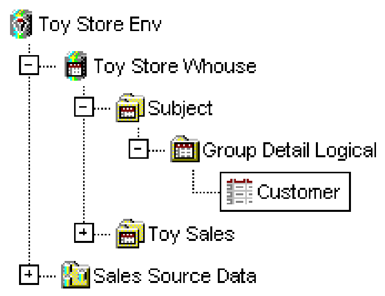
Overview

This example defines a Job for which SAS/Warehouse Administrator generates the code, which means that a Process Flow is required. The Job creates one output table (a Detail Table), with one input source (an ODD).

Note: The following explanations describe the metadata and methods used to achieve the desired results. It is assumed that the appropriate Data Warehouse Environment, Data Warehouse, Subject, Detail Table physical properties, and ODD exist. △

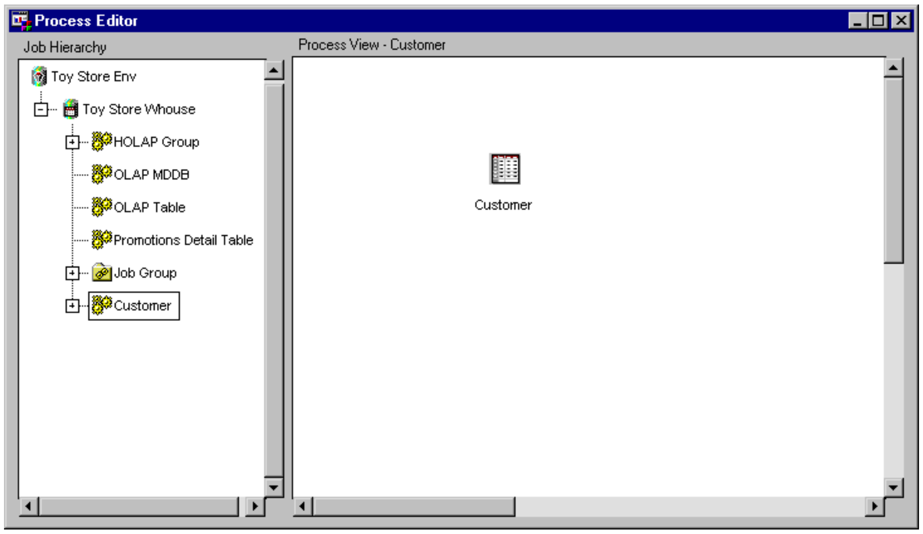
Adding a Job from the Data Store

In the SAS/Warehouse Administrator Explorer, position the cursor on the data store for which you want to add a Job, for example the Detail Table **Customer**.



Right-click and select **Process**. SAS/Warehouse Administrator asks you if you want to create a Job.

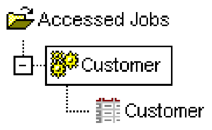
Select **Yes**. The software adds a Job (named by default for the data store), opens the Process Editor window, and displays a Job icon for the output table in the Process View.



Note: By adding the Job using this method, the associated output table is specified automatically for the Job Process Flow, which is displayed in the Process View. △

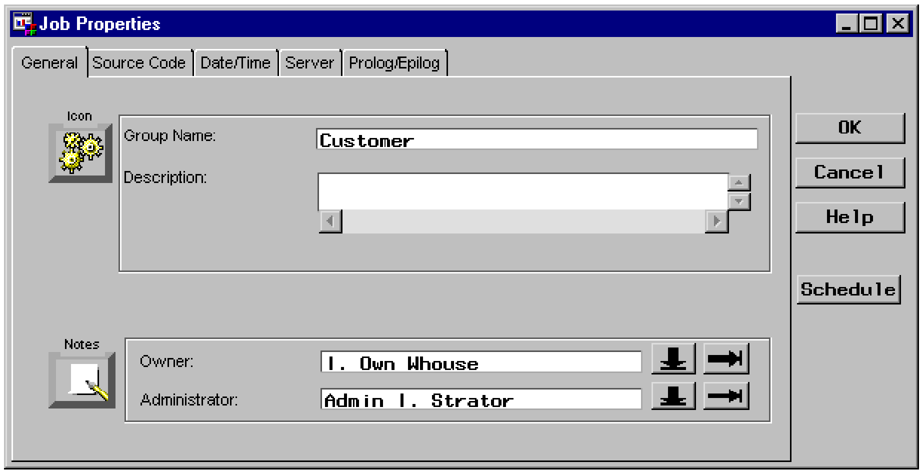
Defining Job Properties

To update the default metadata for the Job, you can start from the Process Editor Job Hierarchy or Job List.



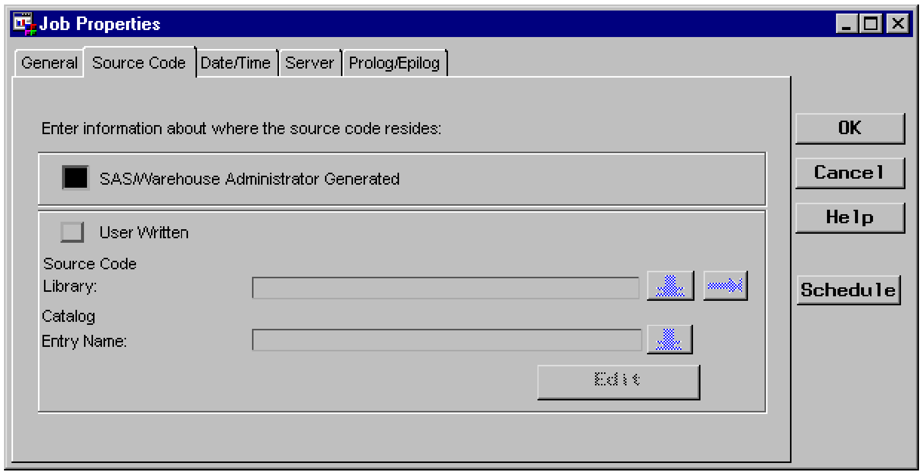
Position the cursor on the Job icon, click the right mouse button, and select **`Properties`**. The Job Properties window displays.

General Tab specifies the Job name, a description, an owner, and an administrator.



Group Name: Customer
Description:
Owner: I. Own Whouse
Administrator: Admin I. Strator

Source Tab specifies who supplies the code for the Job, which for this example is SAS/Warehouse Administrator and is the default setting.



Enter information about where the source code resides:

■ SAS/Warehouse Administrator Generated

☐ User Written

Source Code Library:

Catalog Entry Name:

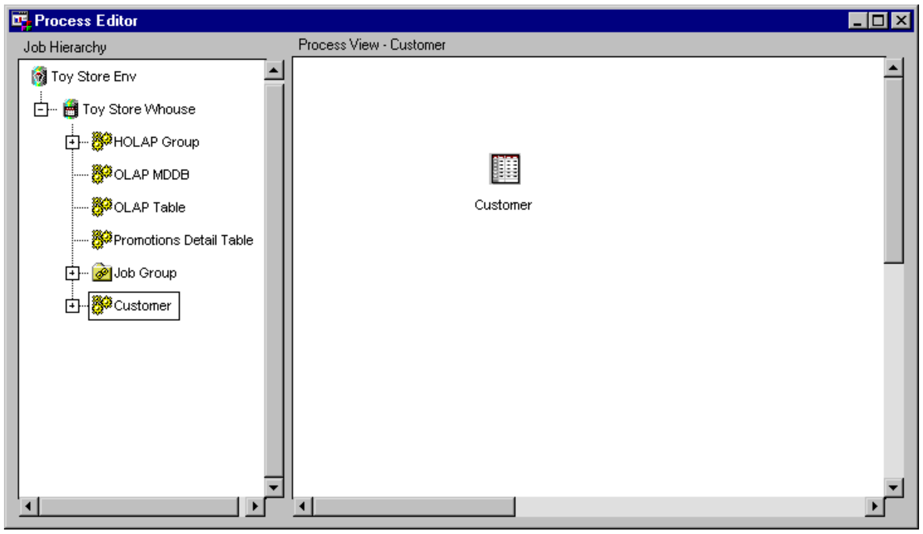
The remaining tabs in the Job Properties window, which are for Job scheduling, are explained in the “Scheduling Jobs” chapter.

Adding an Input Source

After you have an output table specified for a Process Flow, which for this example was specified automatically, you must add its input.

Note: SAS/Warehouse Administrator will not allow you to add an input that is invalid for a particular output table. However, some inputs might be allowed to provide for the possibility of user-written code. △

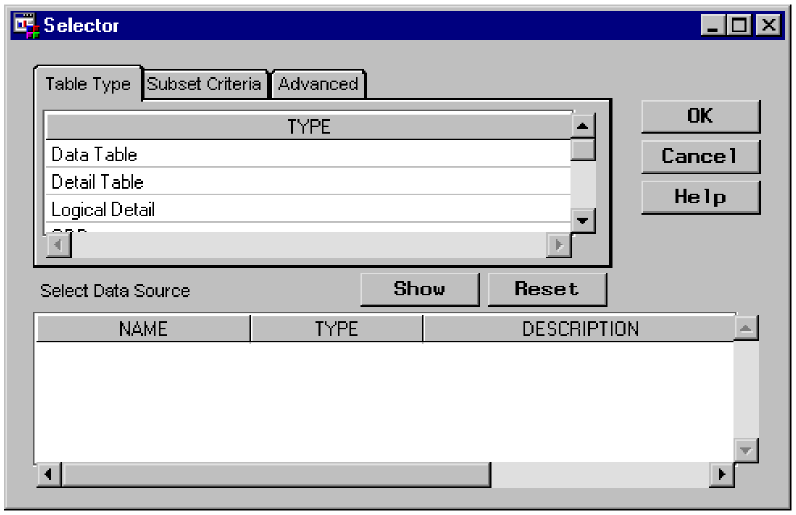
You add an input source to the Job Process Flow in the Process View:



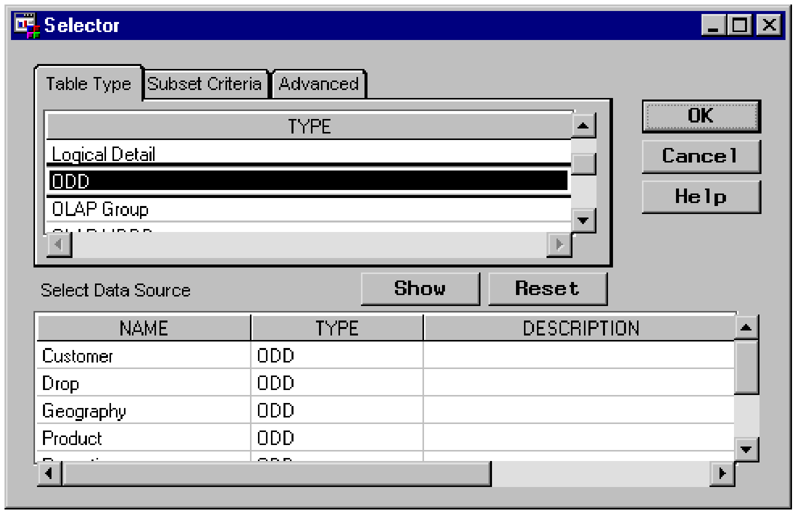
Click the output table with the right mouse button and select

Add ▶ Inputs

The Selector window displays.



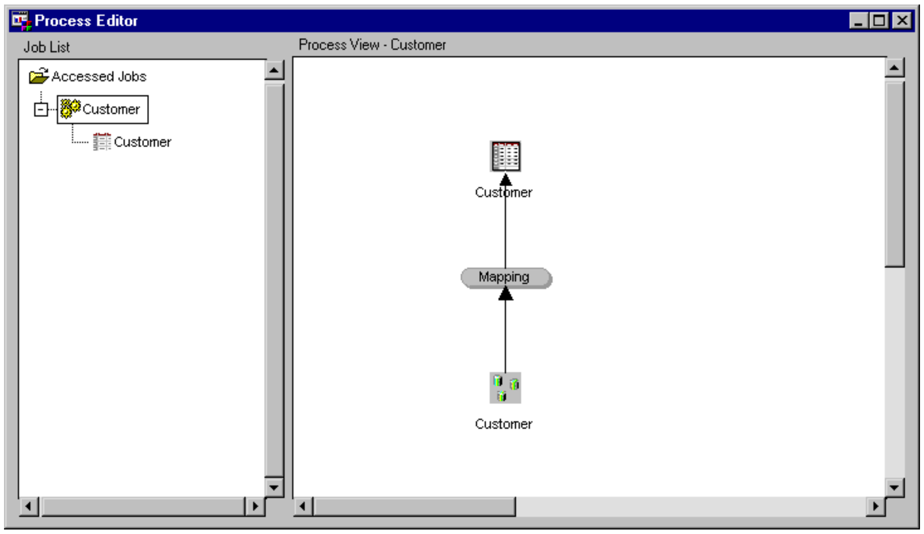
In the Selector window, select the type of table you want to add as an input source, for example **`ODD`**, and then click Show to display the available ODDs.



TYPE
Logical Detail
ODD
OLAP Group

NAME	TYPE	DESCRIPTION
Customer	ODD	
Drop	ODD	
Geography	ODD	
Product	ODD	

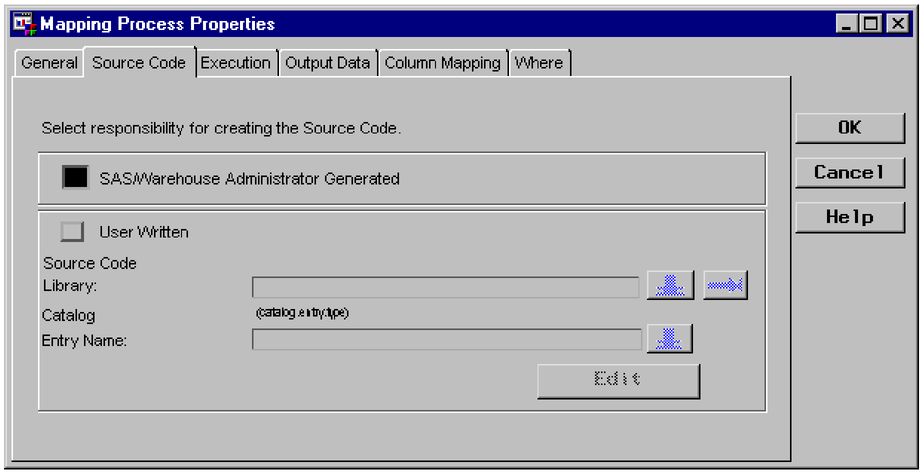
From the displayed list, select the table that you want to specify as an input source, for example **`Customer`**, and then click OK. The input source is added to the Process Flow.



Defining Mapping Process Properties

For most output tables, a default Mapping process will appear between the output table and the input source that was added, which is illustrated in the previous display. The Mapping process maps columns from one or more input sources into one or more output tables.

You must update the default metadata in order for the Mapping process to function properly. To update the default metadata for the Mapping process, position the cursor on the Mapping icon, click the right mouse button, and select **`Properties`**. The Mapping Process Properties window displays.



Select responsibility for creating the Source Code.

■ SAS/Warehouse Administrator Generated

☐ User Written

Source Code Library:

Catalog (catalog.entry.type)

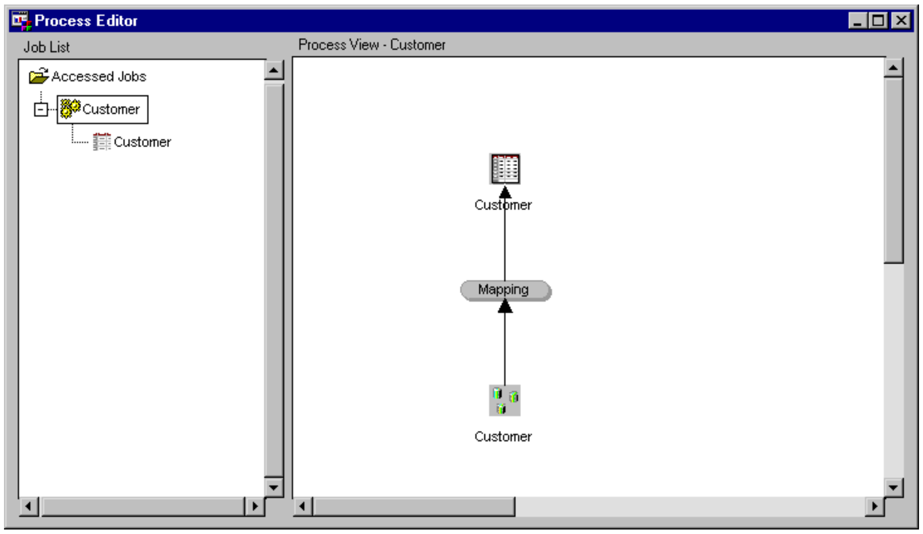
Entry Name:

For an explanation about defining the Mapping process properties, see "Example: Defining Mapping Process Properties for One-to-One Mapping" on page 283 and "Example: Defining Mapping Process Properties to Transform Data" on page 288.

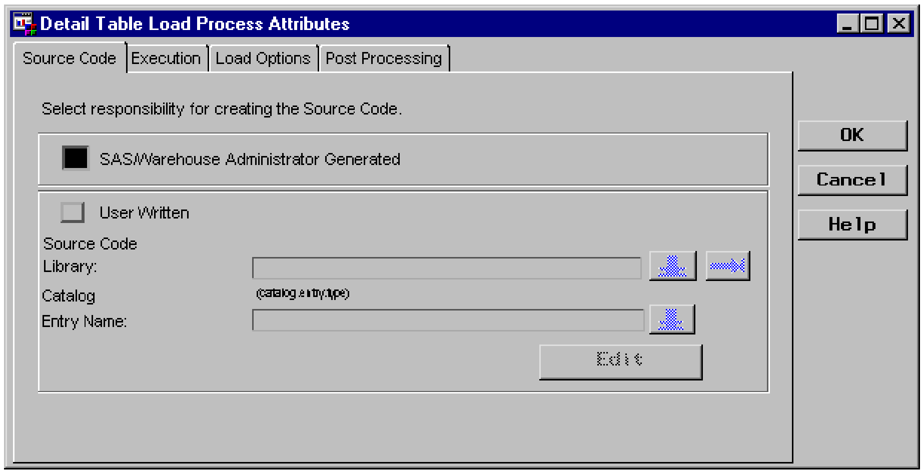
Editing the Load Process

A Load process (also called a Load Step) generates or retrieves code that puts data into a specified output table. The Load process is automatically included in the Process Flow that is associated with the output table. For this example, the Load process default properties do not need to be changed.

However, to update the default metadata for the Load process, position the cursor on the output table in the Process Flow.



Then, click the right mouse button, and select **`Edit Load Step`**. For this example, the Detail Table Load Process Attributes window displays.


Detail Table Load Process Attributes

Select responsibility for creating the Source Code.

■ SAS/Warehouse Administrator Generated

☐ User Written

Source Code Library:

Catalog (catalog.entry.type)

Entry Name:

For an explanation about editing the Load process properties, see "Example: Editing Load Process Properties to Supply User-Written Code" on page 301.

Example: Defining a Job for Which User-Written Code is Supplied

Overview

This example defines a Job for which user-written code is supplied, which means that a Process Flow is not required. The Job creates one output table (a Detail Table), with one input source (an ODD).

Note that when you supply the Job code, you are responsible for allocating any SAS libraries that are referenced in the code. For example, you can include the appropriate LIBNAME statements in the user-written code or in an autoexec file to be executed when SAS is invoked.

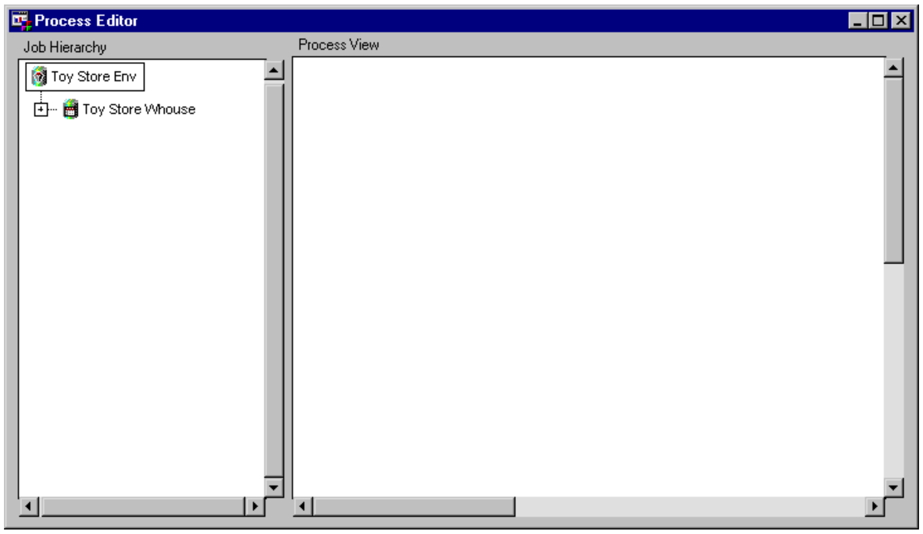
Alternatively, you could use SAS/Warehouse Administrator to generate LIBNAME statements for the input source(s) specified in a Process Flow. That is, you could create a Process Flow even though one is not required because the data store is created with user-written code. The following explanations do not take advantage of this approach, but an example is available in "Define Process Editor Job" on page 124, which illustrates defining a Process Flow for creating an ODD in which user-written code is supplied.

Note: The following explanations describe the metadata and methods used to achieve the desired results. It is assumed that the appropriate Data Warehouse Environment, Data Warehouse, Subject, Detail Table physical properties, and ODD exist. △

Adding a Job

To open the Process Editor, in SAS/Warehouse Administrator Explorer, from the menu bar, select

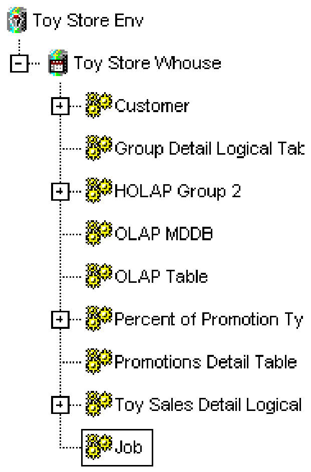
Tools ▶ Process Editor



In the Job Hierarchy, position the cursor on the Data Warehouse for which you want to define a Job, for example **`Toy Store Whouse`**, click the right mouse button, and select

Add ▶ Job

The software adds a Job icon to the Job Hierarchy.

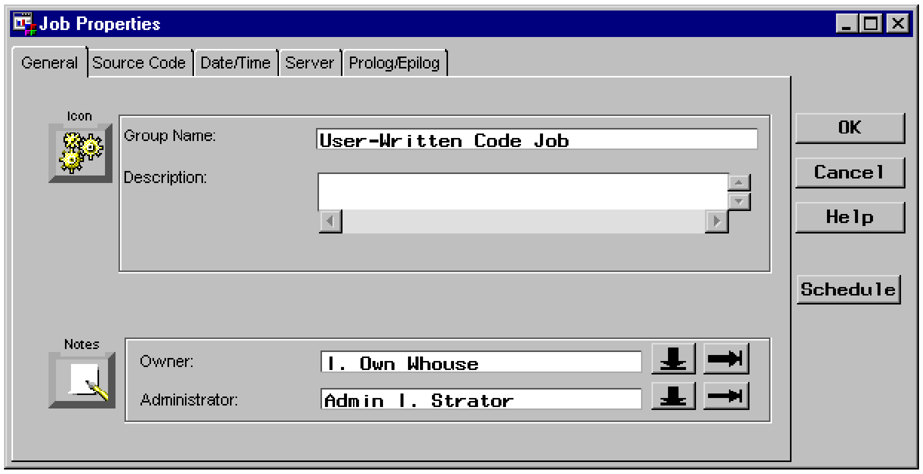


Note: By adding the Job using this method, there is not an output table associated with the Job and therefore there is not a Process Flow. △

Defining Job Properties

To update the default metadata for the Job, position the cursor on the Job icon, click the right mouse button, and select **`Properties`**. The Job Properties window displays for you to enter the appropriate information.

General Tab	specifies the Job name, a description, an owner, and an administrator.



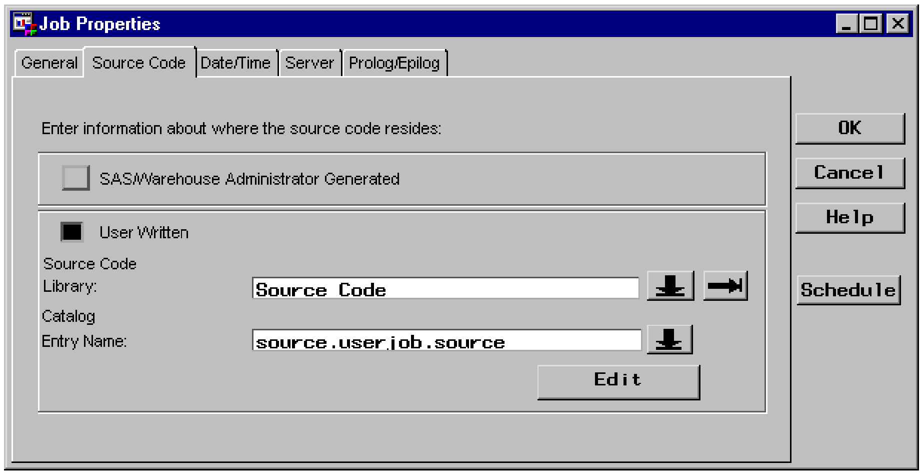
Group Name: User-Written Code Job

Description:

Owner: I. Own Whouse

Administrator: Admin I. Strator

Source Tab specifies that the code for the Job is supplied by user-written code. SAS/Warehouse Administrator will not generate code for the Job. When the **User Written** option is selected, the SAS library and the catalog entry name containing the code must be specified.



Enter information about where the source code resides:

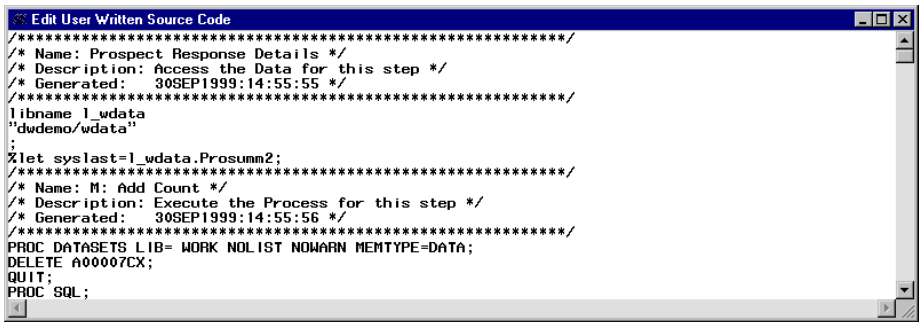
SAS/Warehouse Administrator Generated

User Written

Source Code Library: Source Code

Catalog Entry Name: source.userjob.source

The Edit button opens an editing window where you can enter, view, or update the specified code. Here is an example of code for the Job:



```
/*************************************************************/
/* Name: Prospect Response Details */
/* Description: Access the Data for this step */
/* Generated:   30SEP1999:14:55:55 */
/*************************************************************/
libname l_wdata
"dwdemo/wdata"
;
%let syslast=l_wdata.Prosumm2;
/*************************************************************/
/* Name: M: Add Count */
/* Description: Execute the Process for this step */
/* Generated:   30SEP1999:14:55:56 */
/*************************************************************/
PROC DATASETS LIB= WORK NOLIST NOWARN MEMTYPE=DATA;
DELETE A00007CX;
QUIT;
PROC SQL;
```

The remaining tabs in the Job Properties window, which are for Job scheduling, are explained in the online Help for this window. See also the "Scheduling Jobs" chapter in this document.

Example: Executing a Job

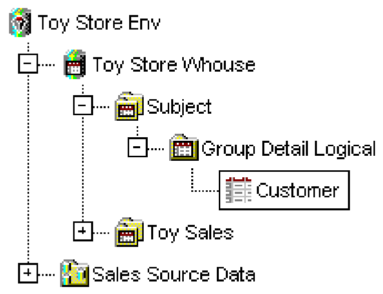
Overview

After you have added a Job and whether you want SAS/Warehouse Administrator to generate the code or you will supply the user-written code yourself, you can execute the Job, which executes the code and loads the output table(s). This example executes a Job interactively. To schedule a Job to run in batch mode at a future date and time, see the "Scheduling Jobs" chapter.

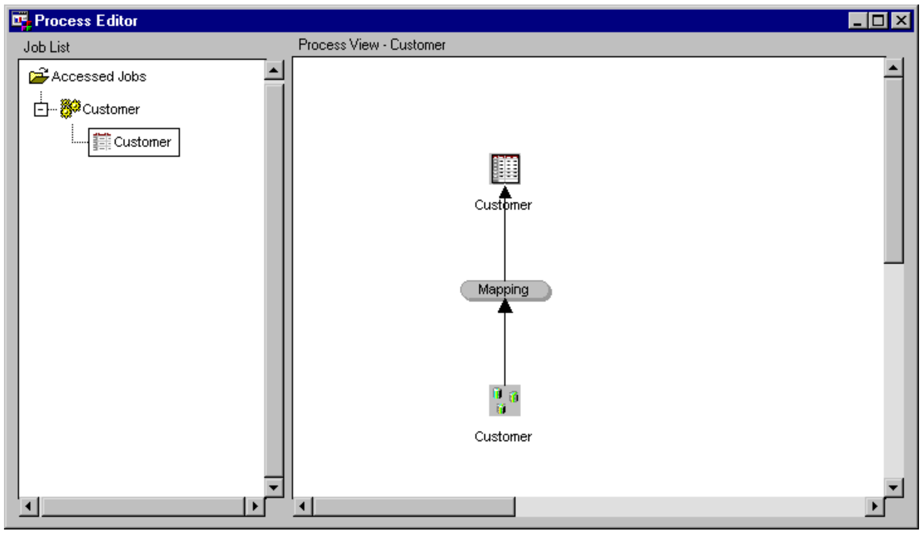
Note: The following explanation describes the metadata and methods used to achieve the desired results and assumes that the appropriate Data Warehouse Environment, Data Warehouse, Subject, data stores, and Job exist. △

Executing a Job Interactively

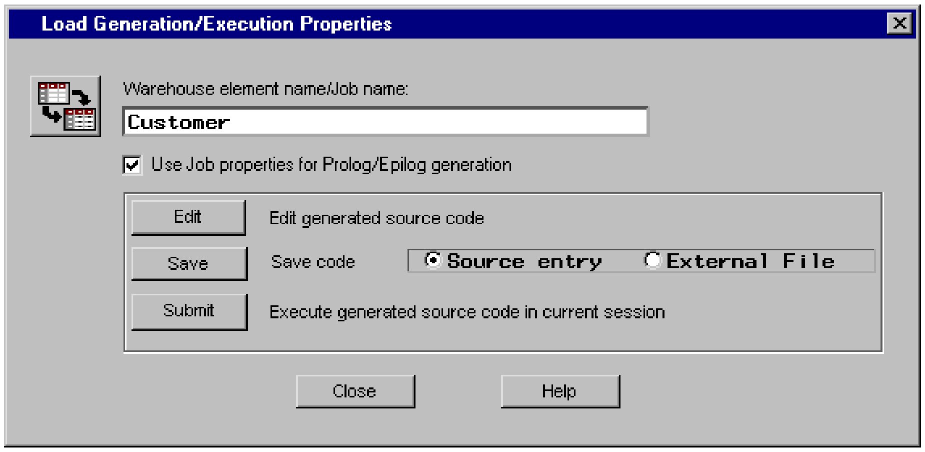
In the SAS/Warehouse Administrator Explorer, position the cursor on the data store for which you want to execute its Job, for example the Detail Table **Customer**.



Click the right mouse button, and select **Process**. The Process Editor opens.



In the Job List, position the cursor on the Job icon, click the right mouse button, and select **`Run`**. The Load Generation/Execution Properties window displays.



Warehouse element name/Job name: Customer

[✓] Use Job properties for Prolog/Epilog generation

Edit generated source code

Save code [● Source entry] ○ External File

Execute generated source code in current session

Click Submit.

Example: Defining a Job with Multiple Output Tables and Input Sources in a Process Flow

Overview

This example defines a Job with a Process Flow that has multiple output tables and multiple input sources. An OLAP Group and both of its children summary data stores are specified in a single job. The Detail Logical Table **`Sales Detail Grp`** is specified as the input source for the two summary data stores as well as the OLAP Group.

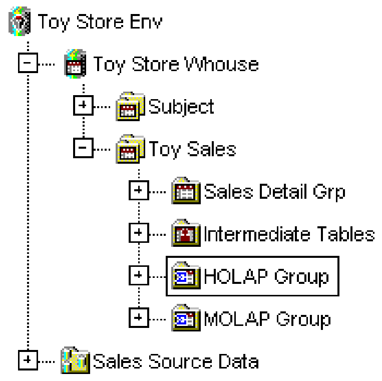
Consider the following when specifying multiple output tables for a Job:

- Only one Job can specify a given data store as an output table, but many Jobs can specify a given data store as an input table.
- If you specify a given data store as an output table and the data store is already specified as the output of another Job, you will be given a series of prompts to determine how the other Job should be handled.
- SAS/Warehouse Administrator will not allow you to add an output table that is invalid for a Job.
- All output tables that you add to a Job must be in the same metadata repository as the Job. For example, you can create a Job whose output tables are all in the same Data Warehouse, and you can create a Job whose output tables are all ODDs in the same Warehouse Environment. But you cannot create a Job in which a Warehouse table and its ODD are both specified as outputs for that Job. For more information, see "Overview: Metadata Repositories" on page 313.

Note: The following explanations describe the metadata and methods used to achieve the desired results. It is assumed that the appropriate Data Warehouse Environment, Data Warehouse, Subject, Detail Logical Table, and OLAP objects physical properties exist. △

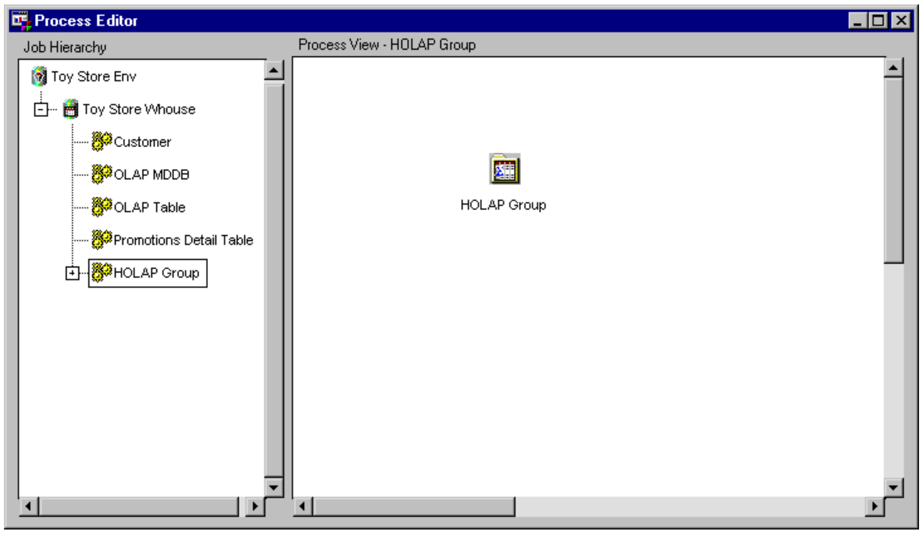
Adding a Job From the Data Store

In the SAS/Warehouse Administrator Explorer, position the cursor on the data store for which you want to add a Job, for example **`HOLAP Group`**.



Click the right mouse button, and select **`Process`**. SAS/Warehouse Administrator asks you if you want to create a Job. Select **`Yes`**.

The software adds a Job (named by default for the data store), opens the Process Editor, and displays an icon for the output table in the Process View.

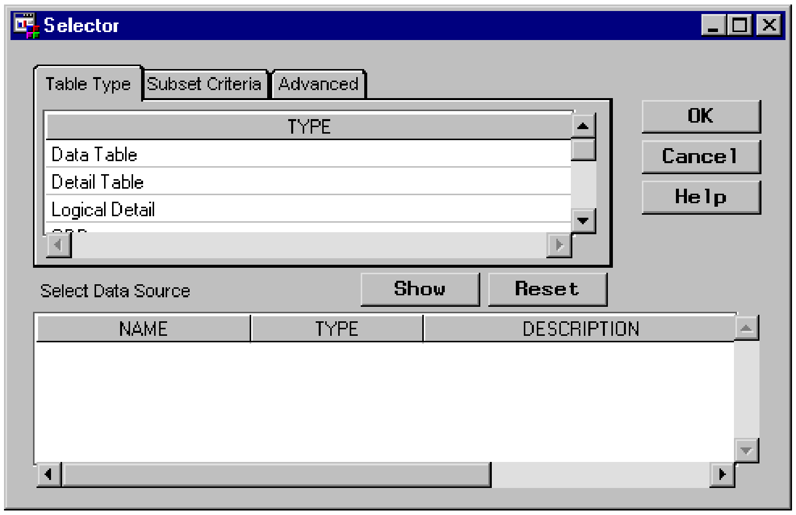


You would now typically define the Job properties, such as a Job name and whether the code for the Job is generated by SAS/Warehouse Administrator or user-written. This example assumes that the Job code is generated by SAS/Warehouse Administrator.

Adding an Input Source

To display the Selector window, in the Process Flow, click the output table **`HOLAP Group`** with the right mouse button and select

$\boxed{\text{Add}}$ ▶ $\boxed{\text{Inputs}}$

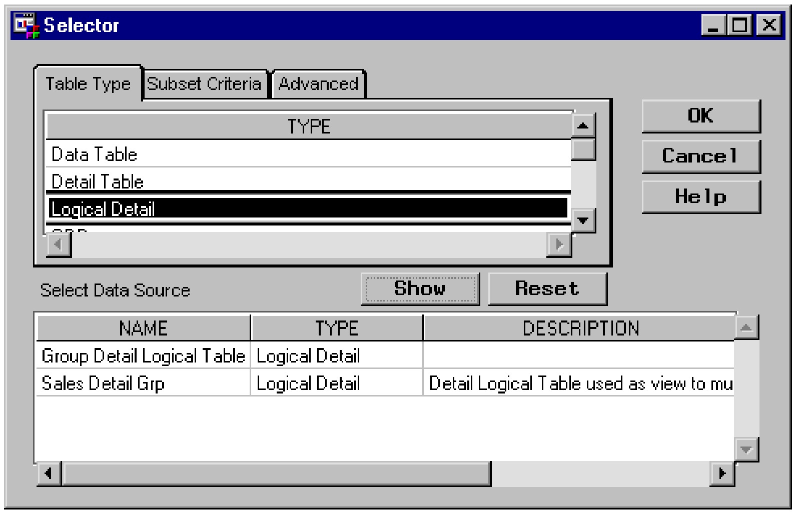


Select Data Source

TYPE
Data Table
Detail Table
Logical Detail

NAME	TYPE	DESCRIPTION

In the Selector window, select the type of table you want to add as an input source, which for this example is a **`Detail Logical Table`**, then click [Show].

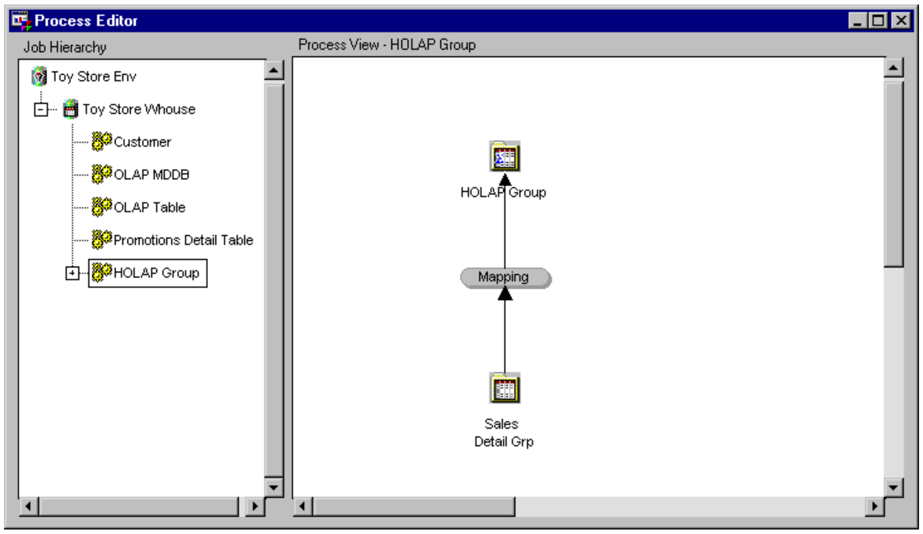


Select Data Source

TYPE
Data Table
Detail Table
Logical Detail

NAME	TYPE	DESCRIPTION
Group Detail Logical Table	Logical Detail	
Sales Detail Grp	Logical Detail	Detail Logical Table used as view to mu

From the list of displayed tables, select the table you want to add, which is **`Sales Detail Grp`**, then click [OK]. The input source is added to the Process Flow.

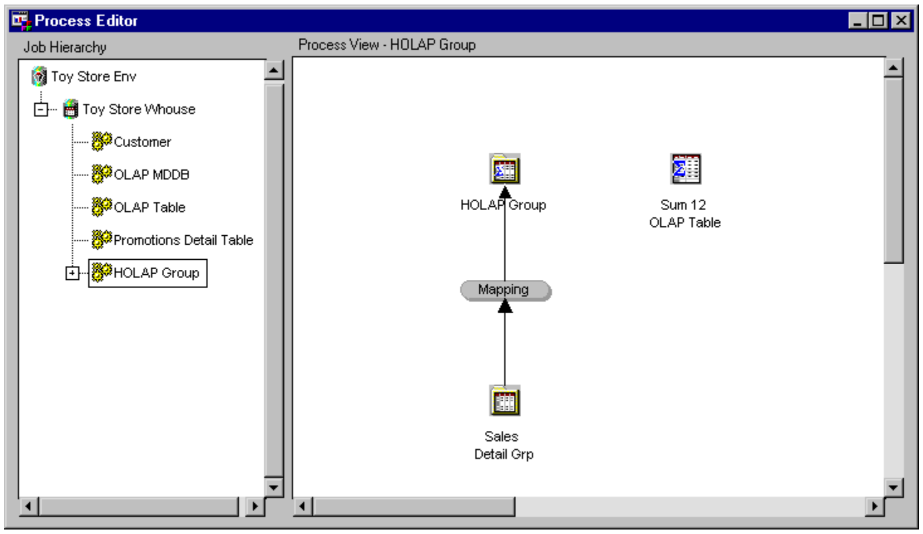


You would now update the default metadata for the Mapping process.

Adding a Second Output Table and Input Source

In the Process Flow, click the right mouse button in background and select **Add Output Table**.

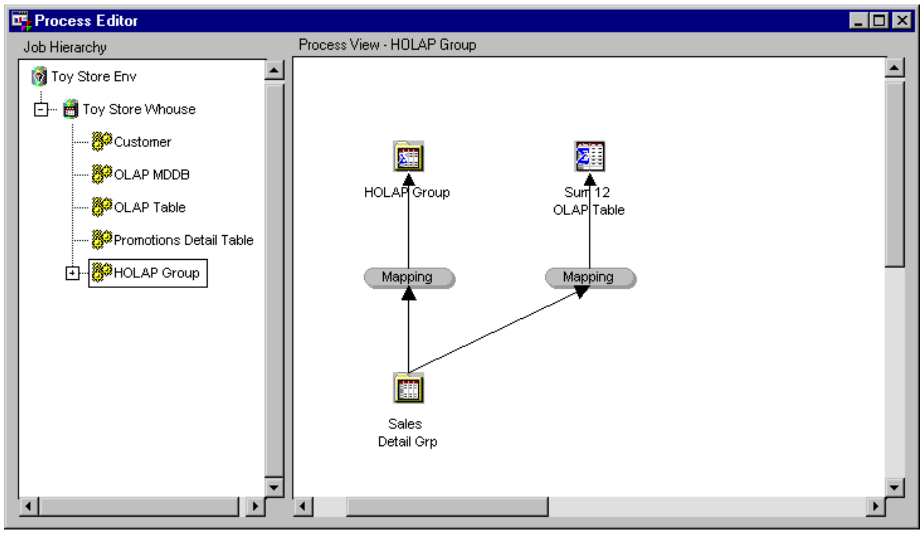
The Selector window displays. In the Selector window, select the type of table you want to add as an additional output table, which for this example is an **OLAP Table**. Then click Show. From the list of displayed tables, select the table you want to add, which is **Sum 12 OLAP Table**, and click OK. The additional output table is added to the Process Flow.



To specify an input source for the additional output table, in the Process Flow, click the output table **Sum 12 OLAP Table** with the right mouse button and select

Add ▶ Inputs

From the Selector window, select the type of table you want to add as an input source, which for this example is a **Detail Logical Table**, then click Show. From the list of displayed tables, select the one you want to add, which is **`Sales Detail Grp`** (the same input source as specified for output table **`HOLAP Group`**), and then click OK. The Process Flow displays that the input source is specified for both output tables.

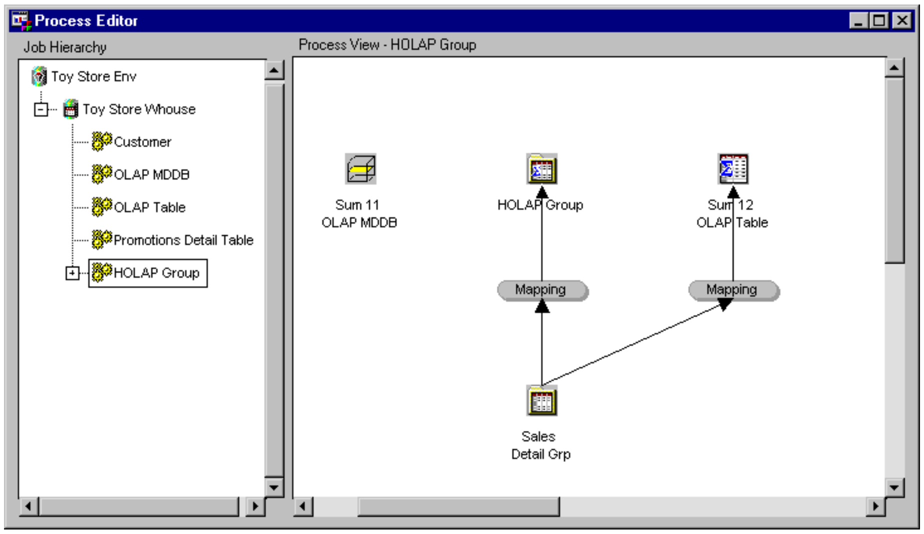


You would now update the default metadata for the Mapping process.

Adding a Third Output Table and Input Source

In the Process Flow, click the right mouse button in background and select **`Add Output Table`**. The Selector window displays.

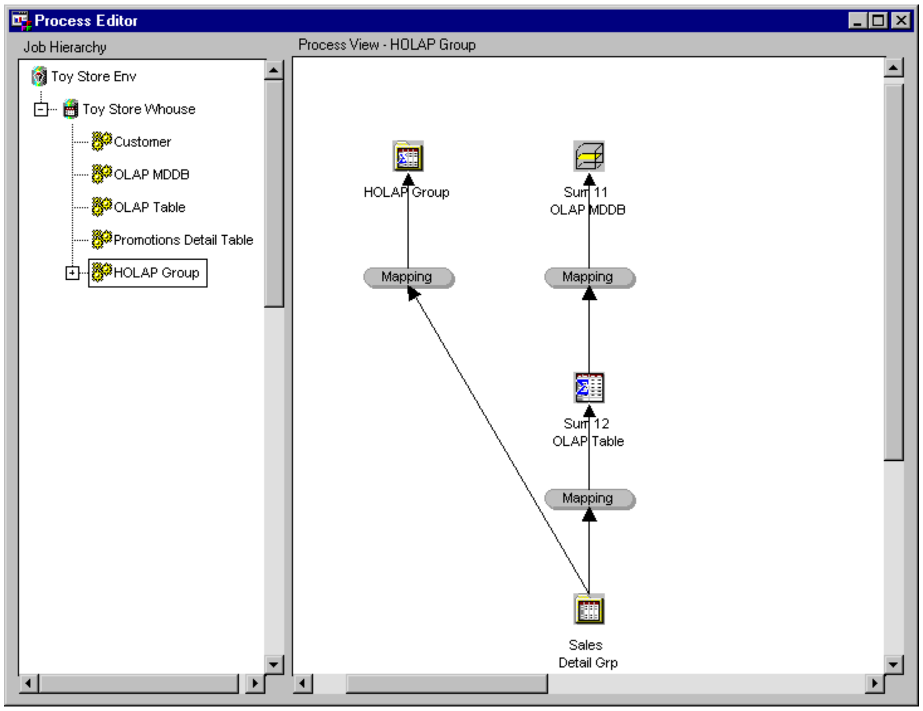
In the Selector window, select the type of table you want to add as an additional output table, which for this example is an **`OLAP MDDB`**. Then click Show. From the list of displayed tables, select the table you want to add, which is **`Sum 11 OLAP MDDB`**, and click OK. The third output table is added to the Process Flow.



To specify an input source for the third output table, in the Process Flow, click the output table with the right mouse button and select

Add ▶ Inputs

From the Selector window, select the type of table you want to add as an input source, which for this example is an **`OLAP Table`**. Then click Show. From the list of displayed tables, select the table you want to add, which is **`Sum 12 OLAP Table`**, then click OK. The input source, which is also an output table in the Process Flow, is displayed in the Process Flow.



You would now update the default metadata for the Mapping process.

Example: Adding a User Exit Process to a Process Flow

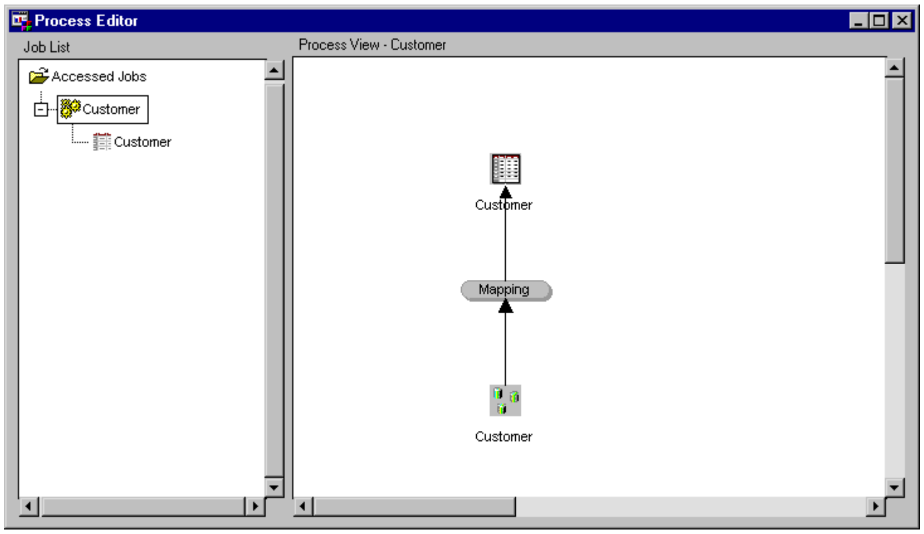
Overview

A User Exit process is a metadata record used to retrieve user-written code. The code often extracts or transforms information for a data store, but it can also do other tasks. The code must be stored in a SAS catalog with an entry type of SOURCE or SCL.

Note: The following explanation describes the metadata and methods used to achieve the desired results. It is assumed that the appropriate Data Warehouse Environment, Data Warehouse, Subject, data stores, Job, and Process Flow exist. △

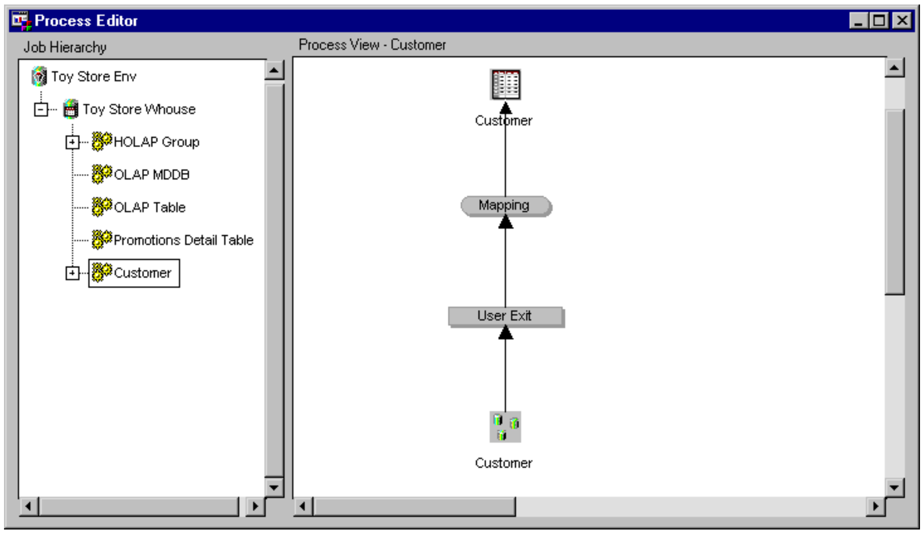
Adding a User Exit Process

In the SAS/Warehouse Administrator Explorer, position the cursor on the data store for which you want to add a User Exit process to its Process Flow, for example **`Customer`**. Click the right mouse button, and select **`Process`**. The software opens the Process Editor with the Process Flow displayed in the Process View.

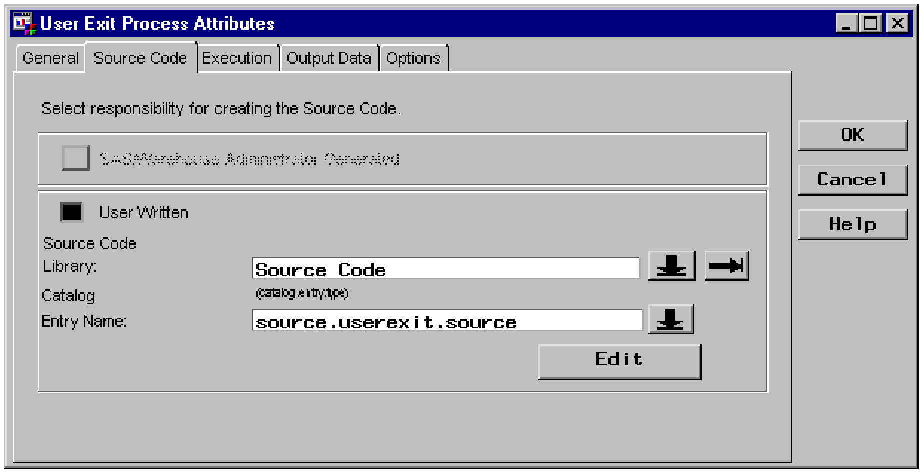


To add a User Exit, in the Process Flow, right-click the Process Flow arrow either below the output table or above the input source, as appropriate, which for this example is the arrow above the input source. Then select

Add ▶ User Exit



To update the default metadata for the User Exit, position the cursor on the User Exit icon, click the right mouse button, and select **`Properties`**. The User Exit Process Attributes window displays for you to enter the appropriate information. Note that the code for a User Exit process is always user-written code and must be stored in a SAS catalog with an entry type of SOURCE or SCL.



Select responsibility for creating the Source Code.

SAS/Warehouse Administrator Generated

User Written

Source Code Library: Source Code

(catalog entry type)

Catalog Entry Name: source.userexit.source

For an explanation about defining the User Exit properties, see "Example: Defining User Exit Process Properties" on page 290.

Example: Adding a Data Transfer Process to a Process Flow

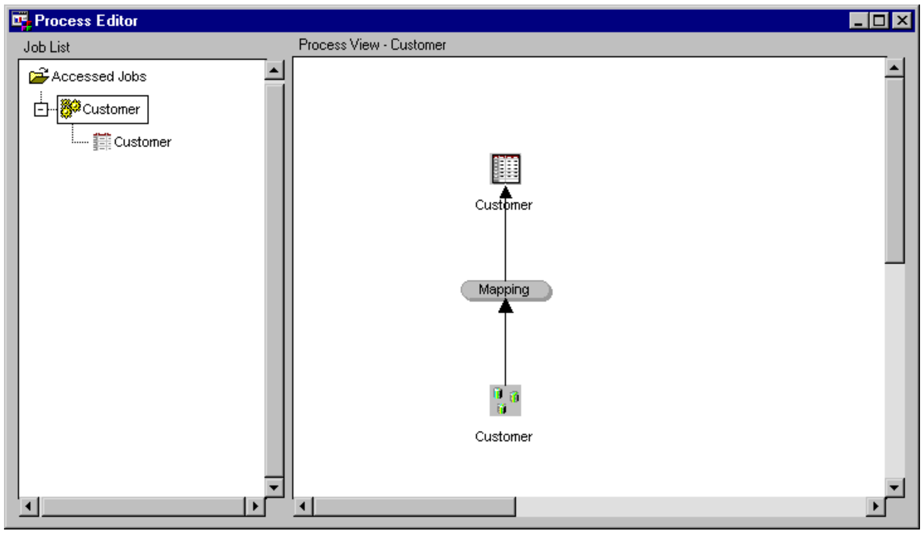
Overview

A Data Transfer process is a metadata record used to generate or retrieve user-written code that moves data from one host to another. For example, a Data Transfer process is required when the output table and its input source reside on different hosts.

Note: The following explanation describes the metadata and methods used to achieve the desired results. It is assumed that the appropriate Data Warehouse Environment, Data Warehouse, Subject, data stores, Job, and Process Flow exist and that the output table and its input source reside on different hosts. △

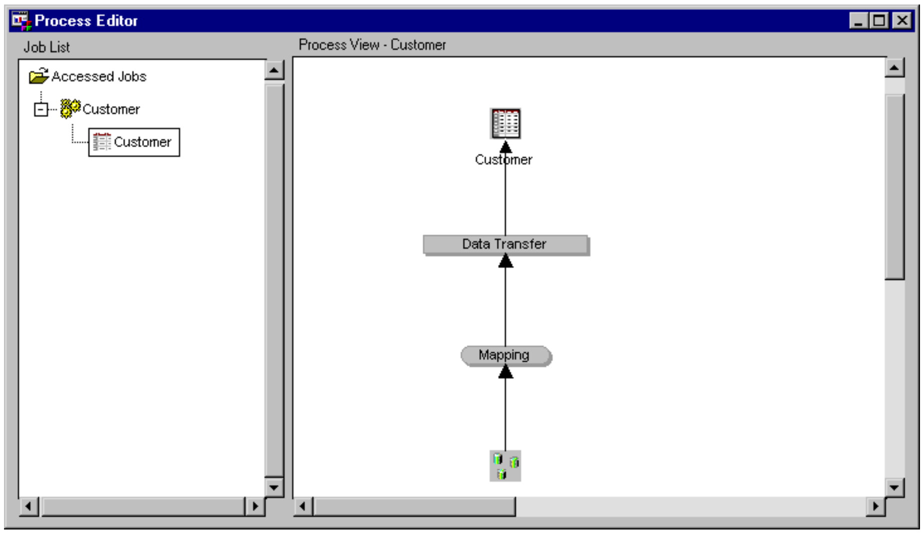
Adding a Data Transfer Process

In the SAS/Warehouse Administrator Explorer, position the cursor on the data store for which you want to add a Data Transfer process to its Process Flow, for example **`Customer`**. Click the right mouse button, and select **`Process`**. The software opens the Process Editor with the Process Flow displayed.

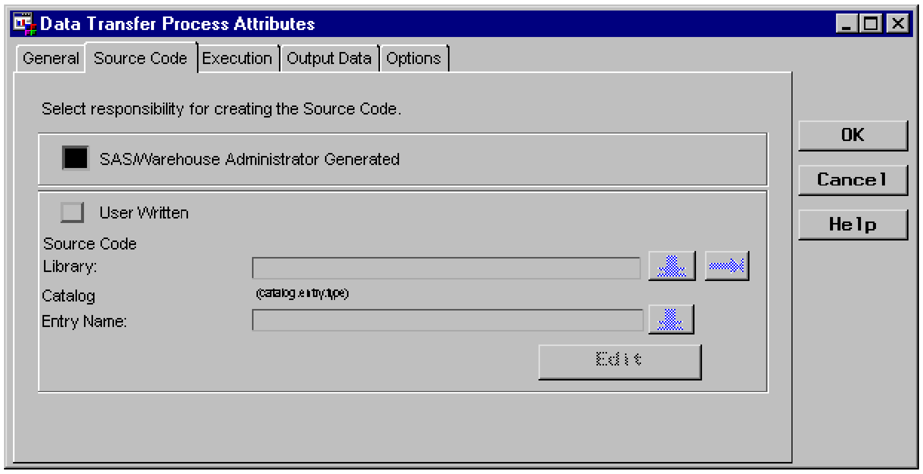


To add a Data Transfer, in the Process Flow, right-click the Process Flow arrow either below the output table or above the input source, as appropriate, which for this example is the arrow below the output table. Then select

Add ▶ Data Transfer



To update the default metadata for the Data Transfer, position the cursor on the Data Transfer icon, click the right mouse button, and select **`Properties`**. The Data Transfer Process Attributes window displays.



Select responsibility for creating the Source Code.

SAS/Warehouse Administrator Generated

User Written

Source Code Library:

Catalog (catalog.entry.type)

Entry Name:

For an explanation about defining the Data Transfer properties, see "Example: Defining Data Transfer Process Properties to Move Data from Remote Host to Local Host" on page 294.

Example: Adding a Record Selector Process to a Process Flow

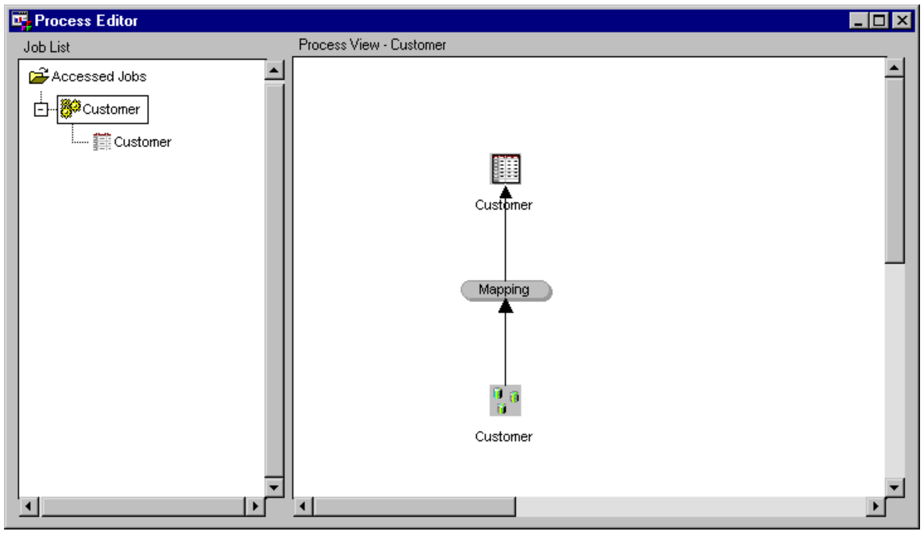
Overview

A Record Selector process is a metadata record used to generate or retrieve user-written code that subsets data prior to loading it to a specified output table. A Record Selector process can be specified only to subset the input source data specified in an ODD or in a Data File (which is an input to an ODD).

Note: The following explanation describes the metadata and methods used to achieve the desired results. It is assumed that the appropriate Data Warehouse Environment, Data Warehouse, Subject, data stores, Job, and Process Flow exist. △

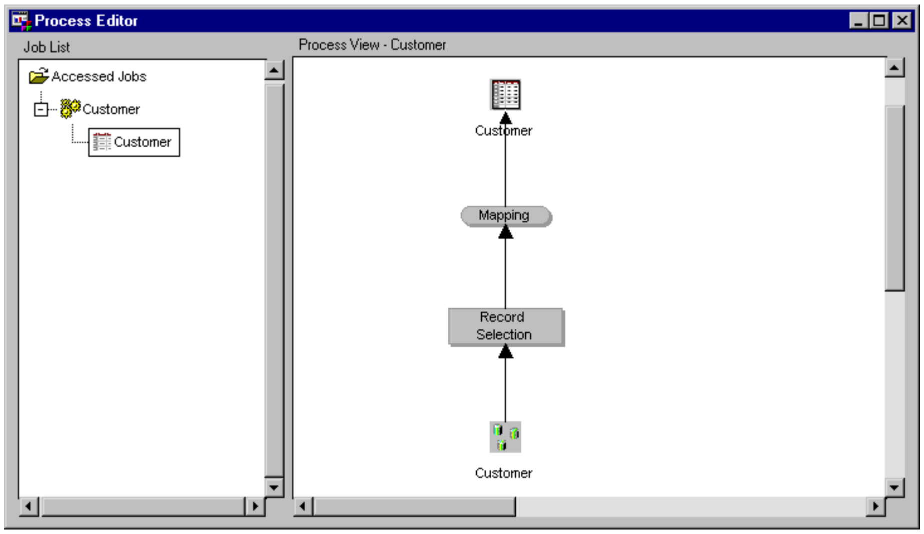
Adding a Record Selector Process

In the SAS/Warehouse Administrator Explorer, position the cursor on the Job to which you want to add a Record Selector process to its Process Flow, for example **`Customer`**. Click the right mouse button, and select **`Process`**. The software opens the Process Editor with the Process Flow displayed.

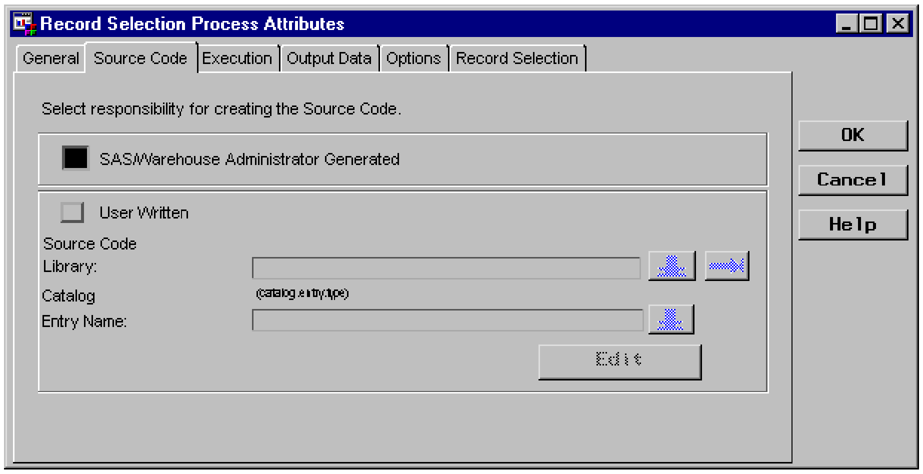


To add a Record Selection, in the Process Flow, right-click the Process Flow arrow above the ODD input source, and select

Add ▶ Record Selector



To update the default metadata for the Record Selector process, position the cursor on the Record Selection icon, click the right mouse button, and select **`Properties`**. The Record Selection Process Attributes window displays.



Select responsibility for creating the Source Code.

■ SAS/Warehouse Administrator Generated

☐ User Written

Source Code Library:

Catalog Entry Name: (catalog.entry.type)

For an explanation about defining the Record Selector properties, see "Example: Defining Record Selector Process Properties" on page 298.

Example: Adding a Job Group

Overview

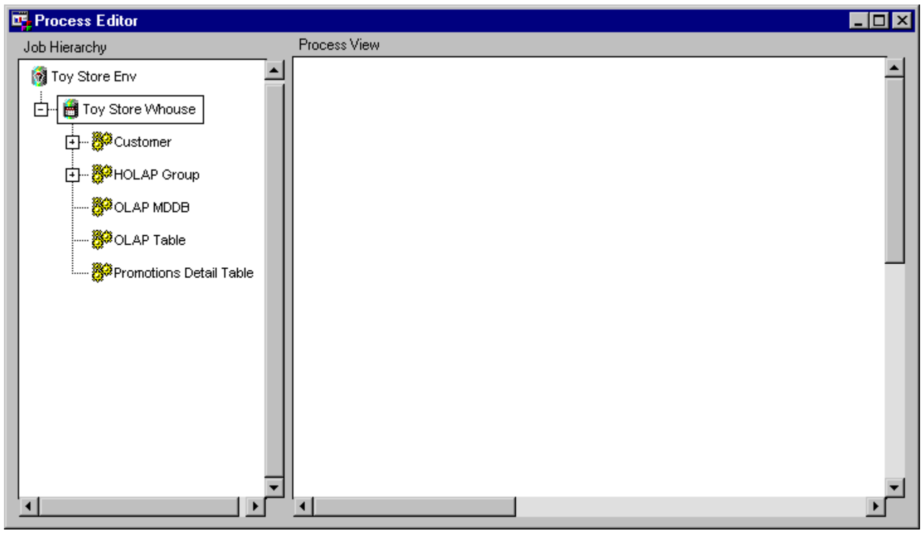
A Job Group is a simple grouping element for Jobs, Events, and other Job Groups. This example adds a Job Group to organize Jobs for the Data Warehouse.

Note: The following explanation describes the metadata and methods used to achieve the desired results. It is assumed that the appropriate Data Warehouse Environment, Data Warehouse, and Subject exist. △

Adding a Job Group

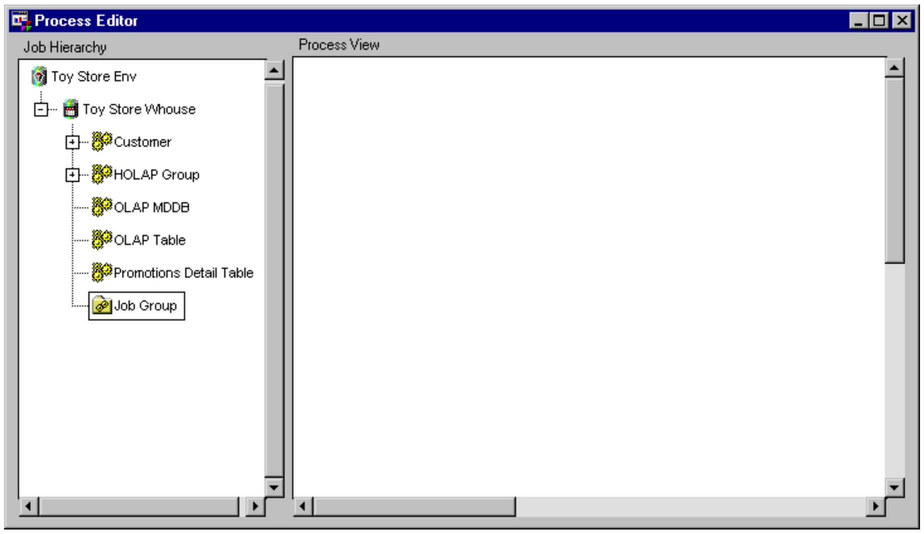
To display the Process Editor, in the SAS/Warehouse Administrator Explorer, from the menu bar, select

Tools ▶ Process Editor



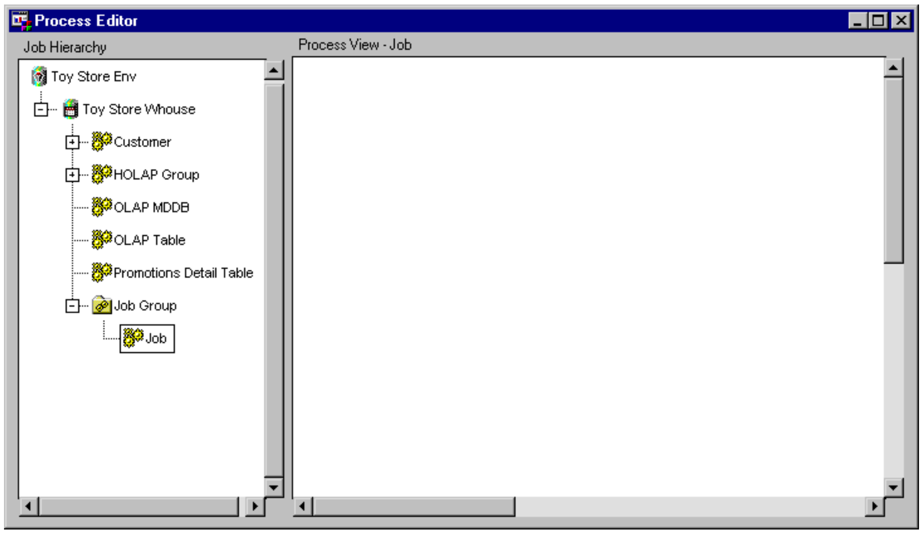
To add a Job Group, in the Job Hierarchy, position the cursor on the Data Warehouse, click the right mouse button, and select

Add ▶ Job Group



To add a Job to the Job Group, position the cursor on the Job Group, click the right mouse button, and select

Add ▶ Job



Example: Moving Jobs

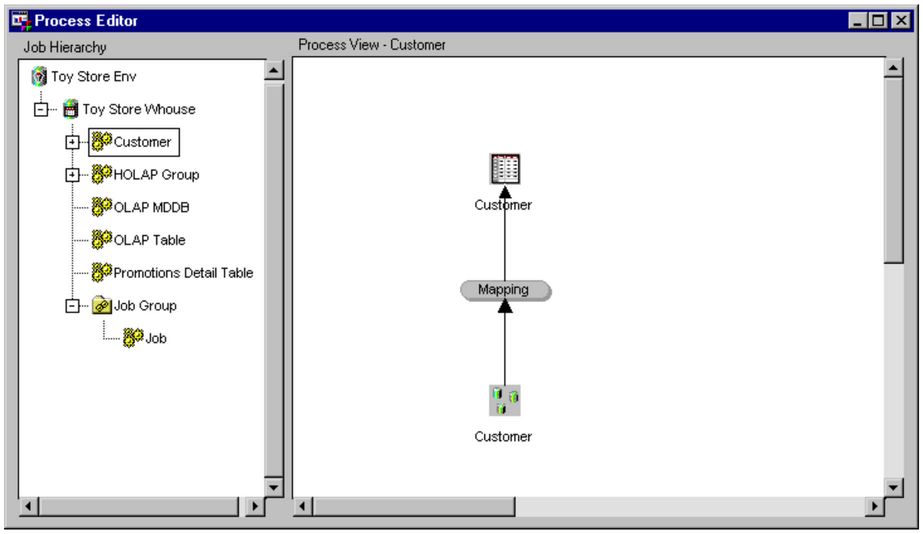
Overview

In the Job Hierarchy, you can move (cut and paste) Job Groups, Jobs, and Events. This allows you to organize these objects in a way that is convenient for later use.

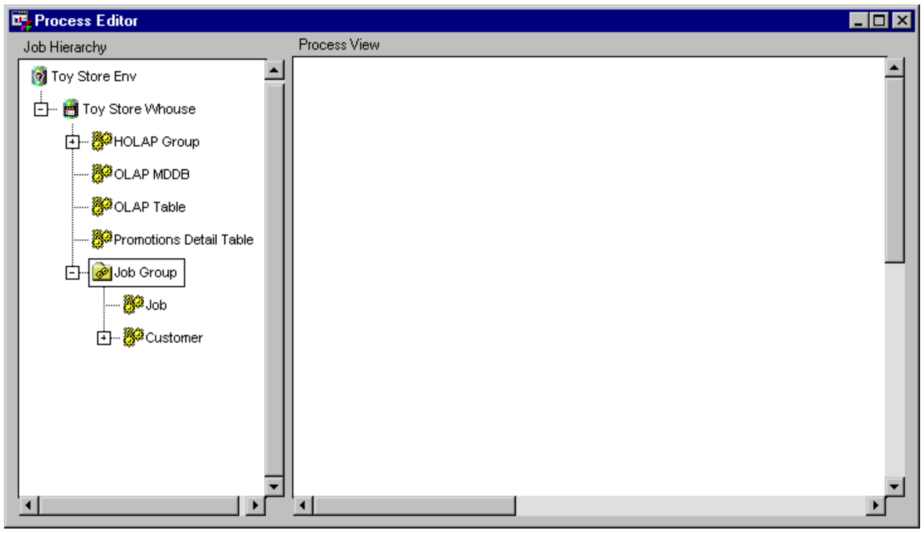
Note: The following explanation describes the metadata and methods used to achieve the desired results. It is assumed that the appropriate Data Warehouse Environment, Data Warehouse, Subject, data store, and Job, and Job Group exist. △

Moving a Job

In the Job Hierarchy, position the cursor on the Job that you want to move, for example **`Customer`**.



Click the right mouse button, and select **`Cut`**. Then, position the cursor on the icon in which to move the Job to, for example **`Job Group`**, click the right mouse button, and select **`Paste`**. The **`Customer`** Job is moved to its new location.



CHAPTER 14 Maintaining Processes

Overview

This chapter assumes that

□ you are familiar with how to define a Job and how to add processes to a Job Process Flow as described in Chapter 13, "Maintaining Jobs," on page 251.

□ you are familiar with the basic information about processes as described in "Processes" on page 52 and are ready to define properties for processes specified in Jobs.

You define properties for the following processes:

Mapping	generates or retrieves code that maps columns from one or more input sources into one or more output tables. Common mapping types include

- □ one-to-one (one input source to an output table).
- □ joins (one or more input sources merged by one or more common columns).
- □ unions (two or more input sources appended to an output table).

User Exit	retrieves user-written code, which, for example, could extract or transform information for a data store.
Data Transfer	generates or retrieves user-written code that moves data from one host to another. A Data Transfer process is required when the output table and its input source reside on different hosts.
Record Selector	generates or retrieves user-written code that subsets data prior to loading it to an output table. In the current release, you can specify a Record Selector process only to subset the input source data from an ODD or a Data File (which is an input to an ODD).
Load	generates or retrieves user-written code that loads data into an output table. A Load process is included automatically in a Process Flow that is associated with the output table.

In general, to maintain processes:

1 Add a process to a Job. See Chapter 13, "Maintaining Jobs," on page 251.
2 Define the properties for a process.

Note: You can access reference information for SAS/Warehouse Administrator process attributes windows by clicking Help. △

User-Written Code

In SAS/Warehouse Administrator, warehouse data stores are created by processes. 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.

In the process attributes window for Load Steps, Mappings, User Exits, Data Transfers, and Record Selectors, on the Source code tab, there is a **User Written** option. Selecting this option enables you to specify a user-written routine to replace the code that SAS/Warehouse Administrator would generate for the process.

For more information about user-written source code, see the online Help. To display the relevant online Help, in the SAS System Help contents, select

Help on SAS Software Products ► Using SAS/Warehouse Administrator Software
► Overview ► Overview of SAS/Warehouse Administrator
► User-Written Source Code

Using the Process Library

SAS/Warehouse Administrator provides a Process Library, which is a collection of registered routines that you can use to extract data, transform it, and load it into a data store. For example, there are routines that standardize addresses, and there are routines that generate the code required to load a warehouse table into a DBMS, such as Oracle, using native loading software.

As you define processes in a Process Flow, you have the option of selecting a predefined routine from the Process Library, rather that supplying user-written code.

The Process Library is made up of a registered set of Process Catalogs. A Process Catalog is a SAS catalog that has a specific set of entries and performs a specific process. The MAIN entry in a Process Catalog is a reference to the routine that actually performs the process.

For more information about the Process Library, click Help on the Process Library window.

Note: Initially, the Process Library is empty. You can install routines as add-in tools from the SAS/Warehouse Administrator Solutions CD, which is provided with SAS/Warehouse Administrator software. △

Example: Defining Mapping Process Properties for One-to-One Mapping

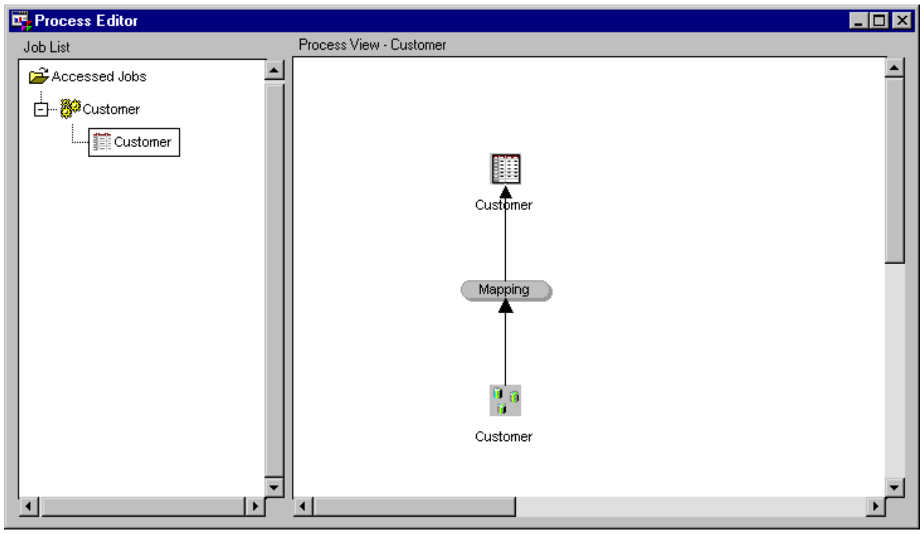
Overview

This example defines the Mapping process properties for a one-to-one column mapping, which is one input source to one output table. The properties also specify that SAS/Warehouse Administrator will generate the Mapping process code.

Note: The following explanation describes the metadata and methods used to achieve the desired results. It is assumed that the appropriate Data Warehouse Environment, Data Warehouse, Subject, data stores, Job, and Process Flow exist. △

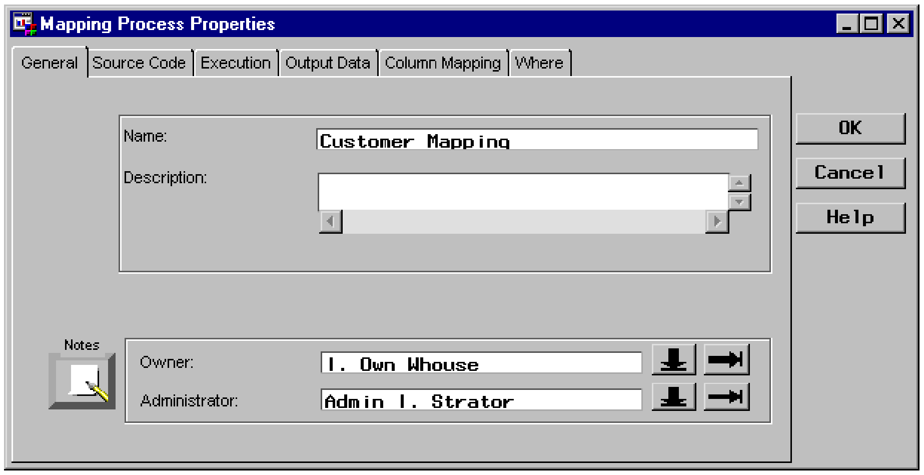
Defining Mapping Properties

For most output tables, a default Mapping process will appear between the output table and an input source after you add the input source to a Job Process Flow as shown in the following Process Flow:



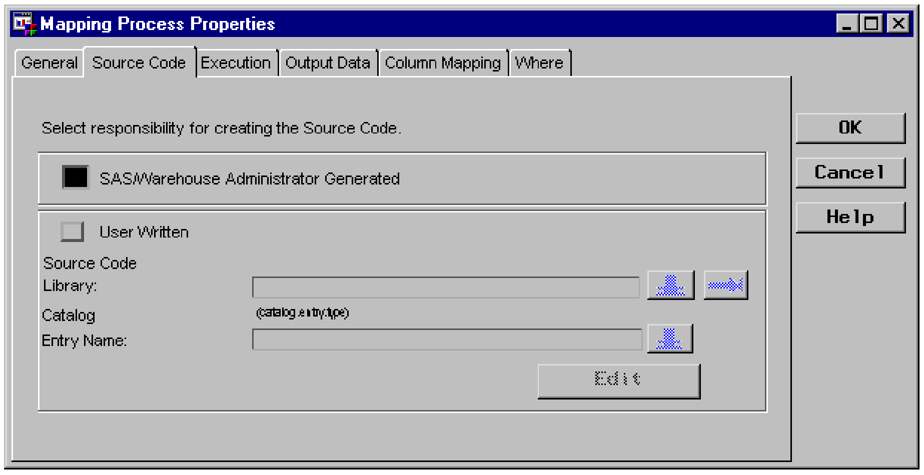
You must always update the default metadata in order for the Mapping process to work. In the Job Process Flow, position the cursor on the Mapping process icon, click the right mouse button, and select **Properties**. The Mapping Process Properties window displays.

General Tab	specifies a name, for example **Customer Mapping**, a description, an owner, and an administrator for the Mapping process.



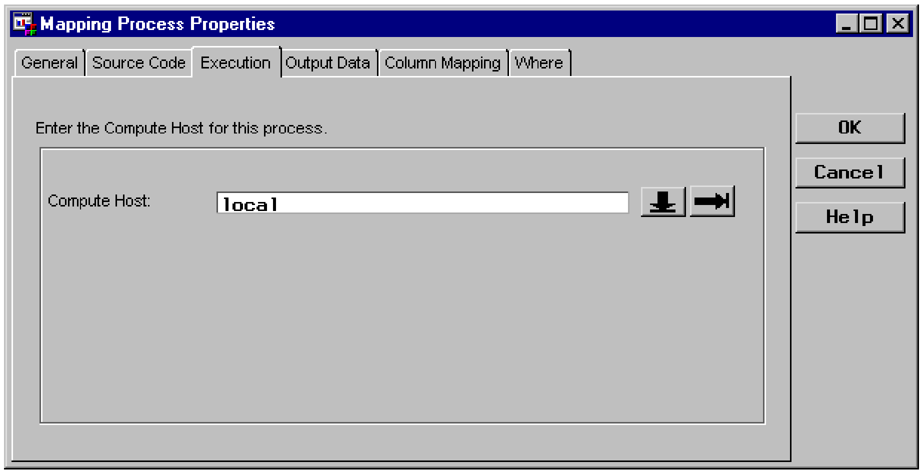
Name: Customer Mapping
Description:
Owner: I. Own Whouse
Administrator: Admin I. Strator

Source Code Tab	specifies who generates the code for the Mapping process, which for this example is SAS/Warehouse Administrator.



Select responsibility for creating the Source Code.
SAS/Warehouse Administrator Generated
User Written
Source Code Library:
(catalog.entry.type)
Catalog Entry Name:

Execution Tab	specifies the host on which you want to run the Mapping process, which for this example is the local host.

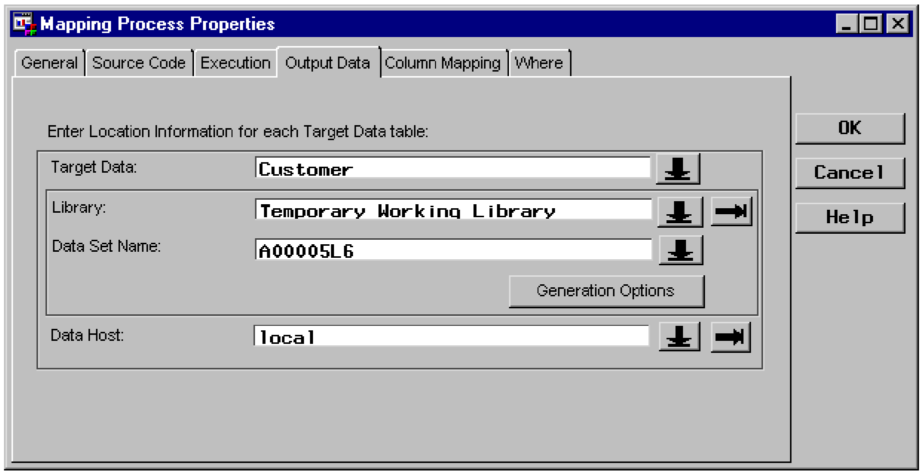


Enter the Compute Host for this process.

Compute Host: local

Output Data Tab

specifies the name and location of the intermediate work table(s) produced by the Mapping process.



Enter Location Information for each Target Data table:

Target Data: Customer

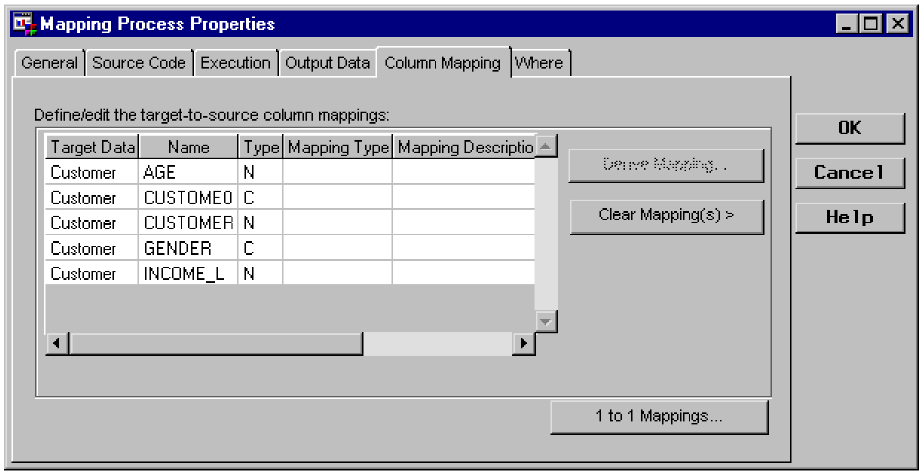
Library: Temporary Working Library

Data Set Name: A00005L6

Data Host: local

Column Mapping Tab

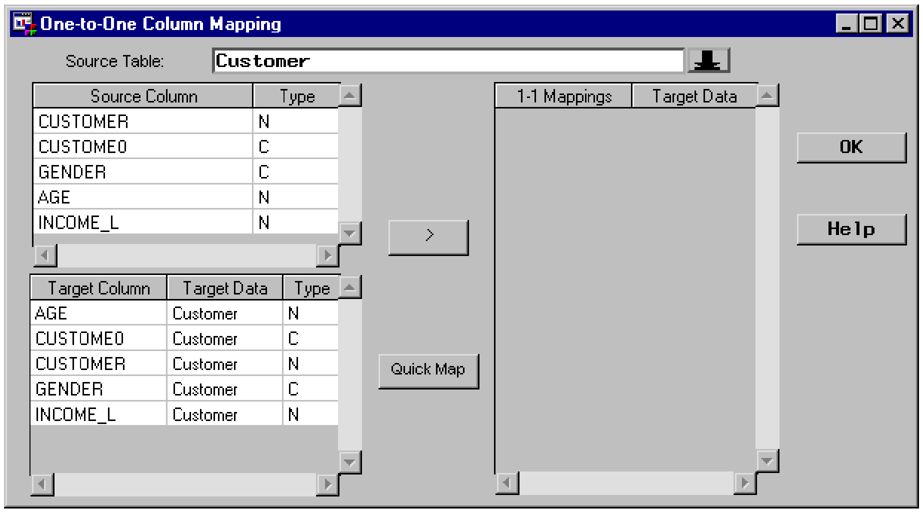
defines how input columns are mapped into the output table(s) and, if specified, how the data in those columns is transformed.



Define/edit the target-to-source column mappings:

Target Data	Name	Type	Mapping Type	Mapping Descriptio
Customer	AGE	N		
Customer	CUSTOMEO	C		
Customer	CUSTOMER	N		
Customer	GENDER	C		
Customer	INCOME_L	N		

To define mappings, you can click either [1 to 1 Mappings], which opens the One-to-One Column Mapping window, or [Derive Mapping], which opens the Expression Builder window. This example selects [1 to 1 Mappings], which enables you to define mappings that do not include data transformations as follows:



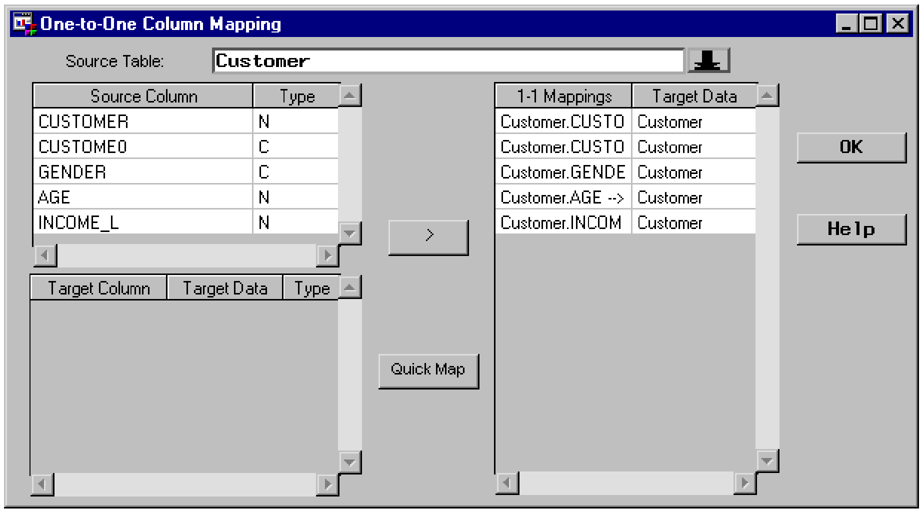
Source Table: Customer

Source Column	Type
CUSTOMER	N
CUSTOMEO	C
GENDER	C
AGE	N
INCOME_L	N

Target Column	Target Data	Type
AGE	Customer	N
CUSTOMEO	Customer	C
CUSTOMER	Customer	N
GENDER	Customer	C
INCOME_L	Customer	N

1-1 Mappings	Target Data

In the One-to-One Column Mapping window, click [Quick Map] and then [OK]. Column names from the source column list are mapped to the target column list and are displayed in the mapping list on the right of the window as follows:



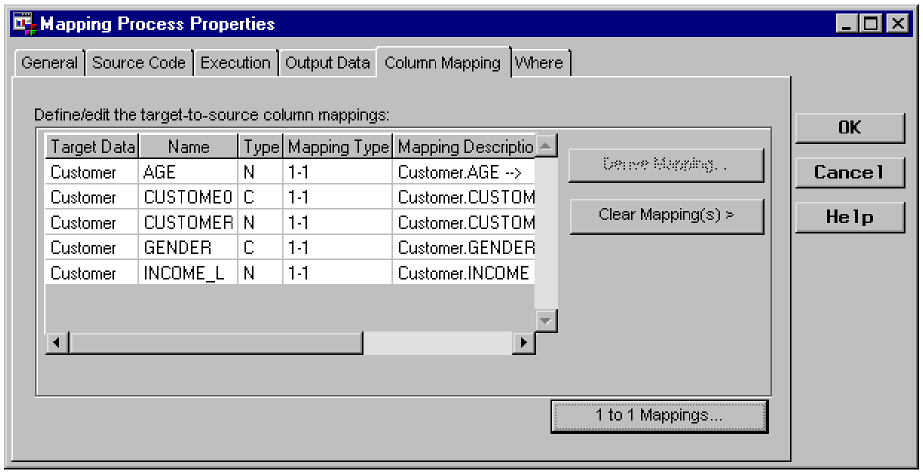
Source Table: Customer

Source Column	Type
CUSTOMER	N
CUSTOMEO	C
GENDER	C
AGE	N
INCOME_L	N

Target Column	Target Data	Type

1-1 Mappings	Target Data
Customer.CUSTO	Customer
Customer.CUSTO	Customer
Customer.GENDE	Customer
Customer.AGE -->	Customer
Customer.INCOM	Customer

Click [OK] to return to the Column Mapping tab in the Mapping Process Properties window, which displays the mapped columns.

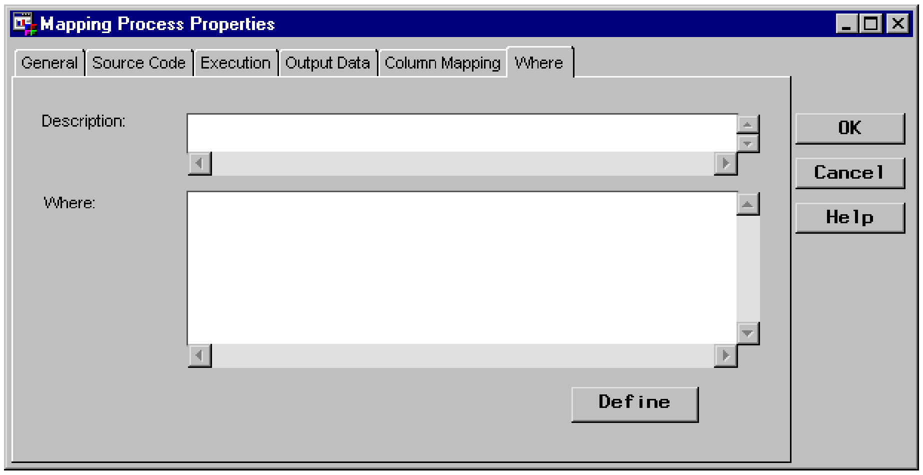

Mapping Process Properties

General | Source Code | Execution | Output Data | Column Mapping | Where

Define/edit the target-to-source column mappings:

Target Data	Name	Type	Mapping Type	Mapping Descriptio
Customer	AGE	N	1-1	Customer.AGE -->
Customer	CUSTOMEO	C	1-1	Customer.CUSTOM
Customer	CUSTOMER	N	1-1	Customer.CUSTOM
Customer	GENDER	C	1-1	Customer.GENDER
Customer	INCOME_L	N	1-1	Customer.INCOME

Where Tab specifies a condition that the data must satisfy before SAS reads it, which for this example is not specified.

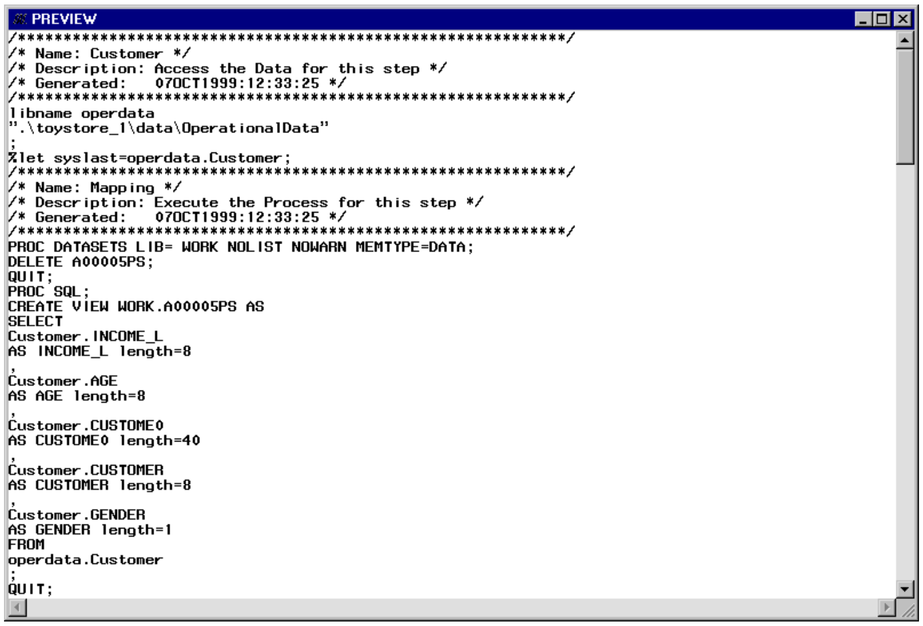

Mapping Process Properties

General | Source Code | Execution | Output Data | Column Mapping | Where

Description:

Where:

The following window shows some of the generated code for the Mapping process:



PREVIEW

```
/*************************************************************/
/* Name: Customer */
/* Description: Access the Data for this step */
/* Generated:   07OCT1999:12:33:25 */
/*************************************************************/
libname operdata
".\toystore_1\data\OperationalData"
;
%let syslast=operdata.Customer;
/*************************************************************/
/* Name: Mapping */
/* Description: Execute the Process for this step */
/* Generated:   07OCT1999:12:33:25 */
/*************************************************************/
PROC DATASETS LIB= WORK NOLIST NOWARN MEMTYPE=DATA;
DELETE A00005PS;
QUIT;
PROC SQL;
CREATE VIEW WORK.A00005PS AS
SELECT
Customer.INCOME_L
AS INCOME_L length=8

,
Customer.AGE
AS AGE length=8

,
Customer.CUSTOME0
AS CUSTOME0 length=40

,
Customer.CUSTOMER
AS CUSTOMER length=8

,
Customer.GENDER
AS GENDER length=1
FROM
operdata.Customer
;
QUIT;
```

Example: Defining Mapping Process Properties to Transform Data

Overview

This example defines the Mapping process properties to transform data by specifying a derived mapping. Only the Column Mapping tab for the Mapping Process Properties window is illustrated.

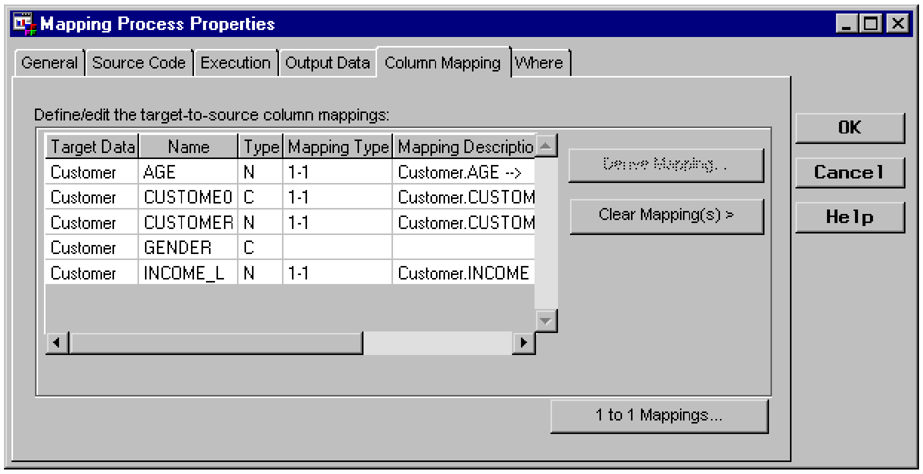
For this example, the input source is an ODD named **`Customer`**, and the output table is a Detail Table named **`Custdet`**. Both tables have a column named **`Gender`**. The ODD **`Gender`** is a numeric column that stores the values **`0`** or **`1`**. The Detail Table **`Gender`** is a character column that stores the values **`Male`** or **`Female`**.

Note: The following explanation describes the metadata and methods used to achieve the desired results. It is assumed that the appropriate Data Warehouse Environment, Data Warehouse, Subject, data stores, Job, and Process Flow exist. △

Defining Mapping Properties

In the Job Process Flow, position the cursor on the Mapping process icon, click the right mouse button, and select **`Properties`**. The Mapping Process Properties window displays. Only the Column Mapping tab, which defines the transformation, is explained here:

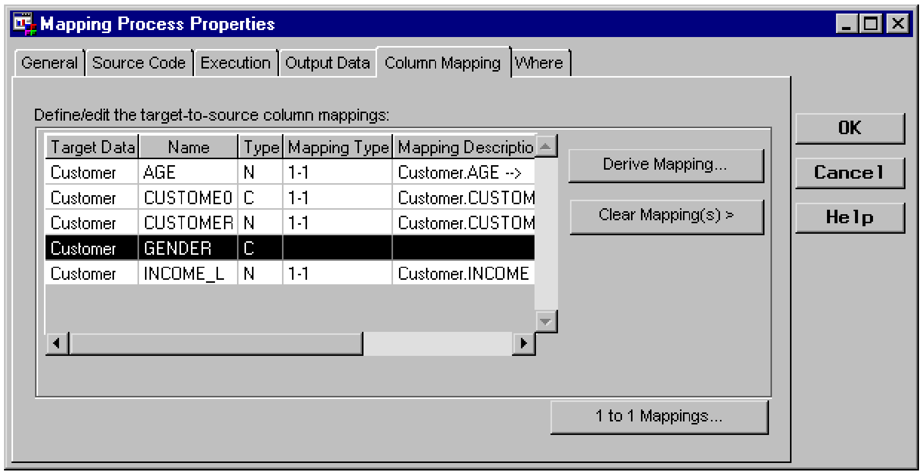
Column Mapping Tab	defines how input columns are mapped into the output table(s) and how the data in those columns is transformed. This example transforms data by specifying a derived mapping for the **`Gender`** column.



Define/edit the target-to-source column mappings:

Target Data	Name	Type	Mapping Type	Mapping Descriptio
Customer	AGE	N	1-1	Customer.AGE -->
Customer	CUSTOMER0	C	1-1	Customer.CUSTOM
Customer	CUSTOMER	N	1-1	Customer.CUSTOM
Customer	GENDER	C		
Customer	INCOME_L	N	1-1	Customer.INCOME

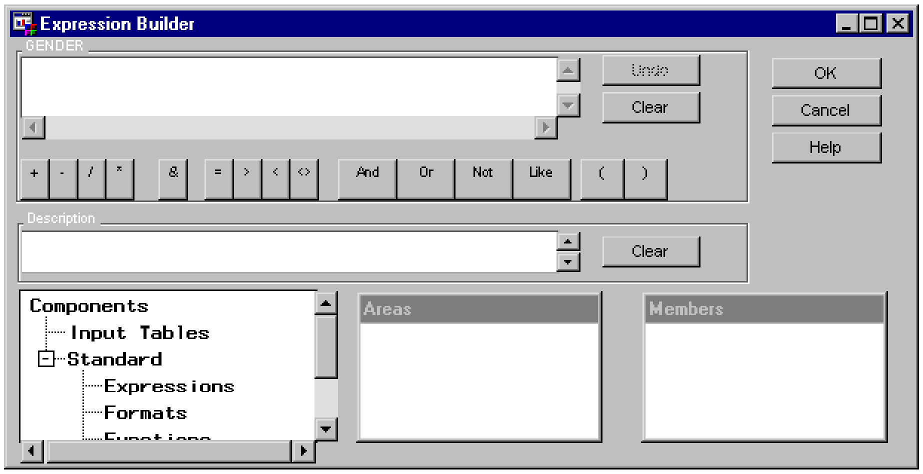
First, select the column to be transformed:



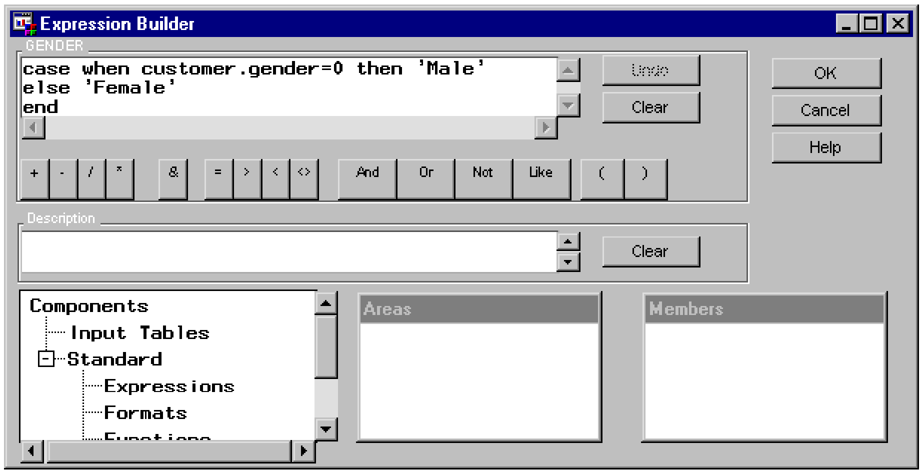
Define/edit the target-to-source column mappings:

Target Data	Name	Type	Mapping Type	Mapping Descriptio
Customer	AGE	N	1-1	Customer.AGE -->
Customer	CUSTOMER0	C	1-1	Customer.CUSTOM
Customer	CUSTOMER	N	1-1	Customer.CUSTOM
Customer	GENDER	C		
Customer	INCOME_L	N	1-1	Customer.INCOME

Then, click Derive Mapping, which opens the Expression Builder window:

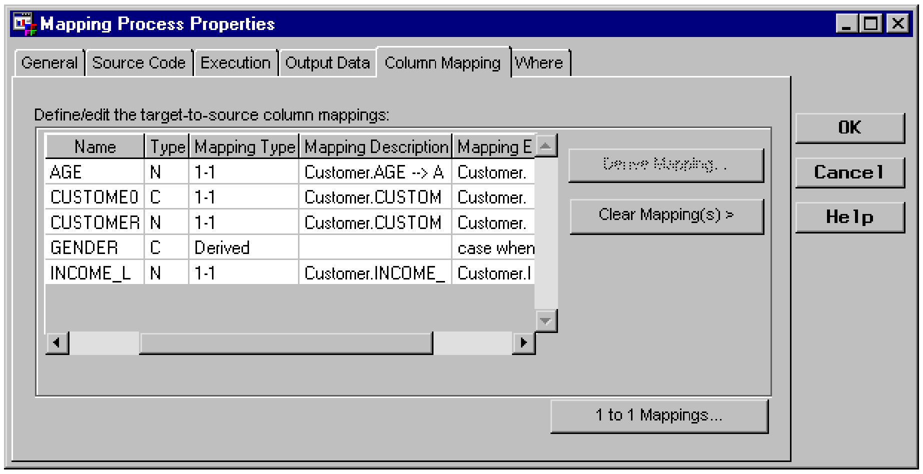


Using the Expression Builder window, you can define a SAS expression, which can transform columns, provide conditional processing, calculate new values, and assign new values. To define an expression, position the cursor in the Transformation field, which automatically displays the name of the selected column, and then select from the Components list and operator buttons. Or, you can enter the appropriate expression. For example, the following expression would transform the numeric values **0** and **1** (stored in the ODD) to the character values **Male** and **Female** (for the Detail Table):



```
case when customer.gender=0 then 'Male'
else 'Female'
end
```

Click [OK] to return to the Column Mapping tab, which displays the derived mapping.



Define/edit the target-to-source column mappings:

Name	Type	Mapping Type	Mapping Description	Mapping E
AGE	N	1-1	Customer.AGE --> A	Customer.
CUSTOMEO	C	1-1	Customer.CUSTOM	Customer.
CUSTOMER	N	1-1	Customer.CUSTOM	Customer.
GENDER	C	Derived		case when
INCOME_L	N	1-1	Customer.INCOME_	Customer.I

Example: Defining User Exit Process Properties

Overview

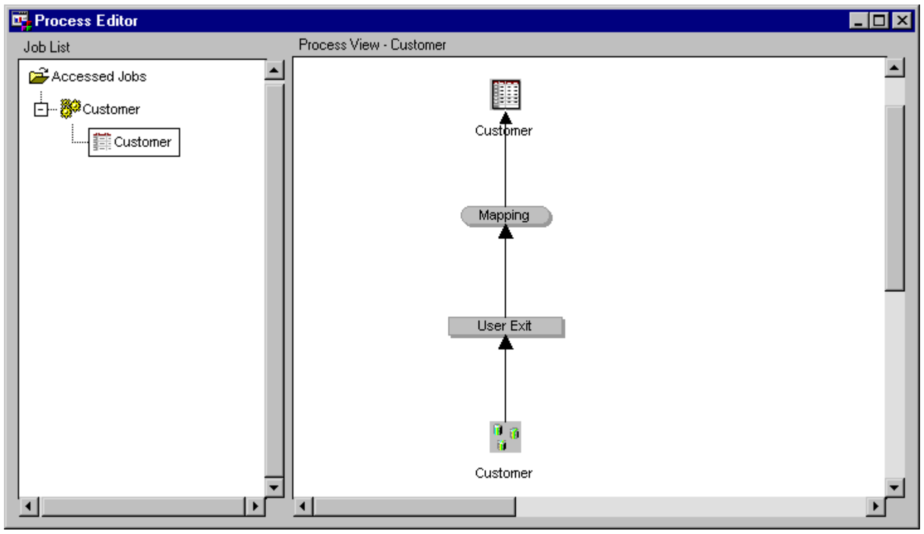
This example defines the User Exit process properties to specify the location of user-written code, which for this example transforms data from the input source before it is loaded into the output table.

For an example about adding a User-Exit process, see “Example: Adding a User Exit Process to a Process Flow” on page 270.

Note: The following explanation describes the metadata and methods used to achieve the desired results. It is assumed that the appropriate Data Warehouse Environment, Data Warehouse, Subject, data stores, Job, and Process Flow exist. △

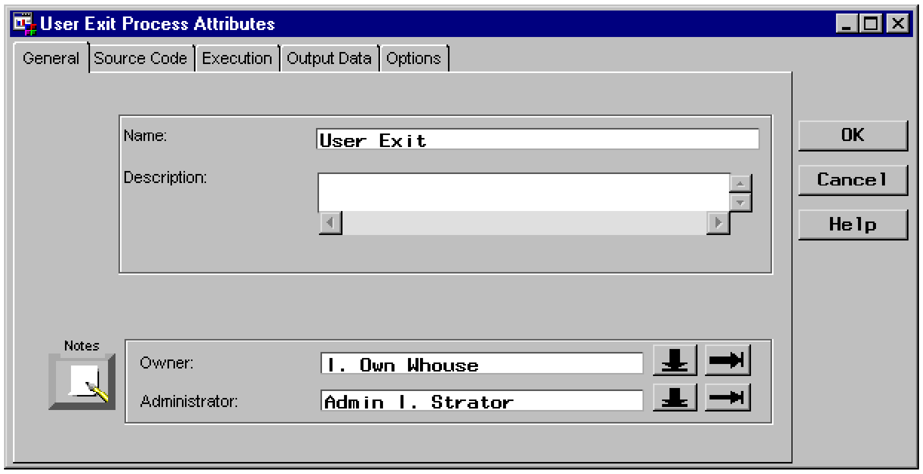
Defining User Exit Properties

In a Job Process Flow, a User Exit process can be added either below the output table or above the input source as appropriate. For this example, the User Exit process was added above the input source as shown in the following Process Flow:



In the Job Process Flow, position the cursor on the User Exit process icon, click the right mouse button, and select **`Properties`**. The User Exit Process Attributes window displays.

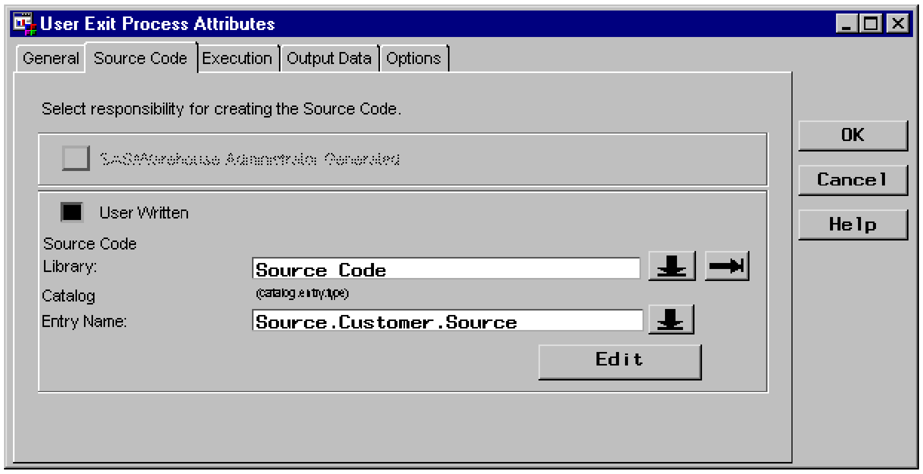
General Tab	specifies a name, a description, an owner, and an administrator for the User Exit process.



Name: User Exit
Description:
Owner: I. Own Whouse
Administrator: Admin I. Strator

Source Code Tab	specifies that the source code for the process is supplied by user-written code, which is required for a User Exit, and specifies

the name of the SAS library that contains the code. The catalog entry name must be either SOURCE or SCL.



Select responsibility for creating the Source Code.

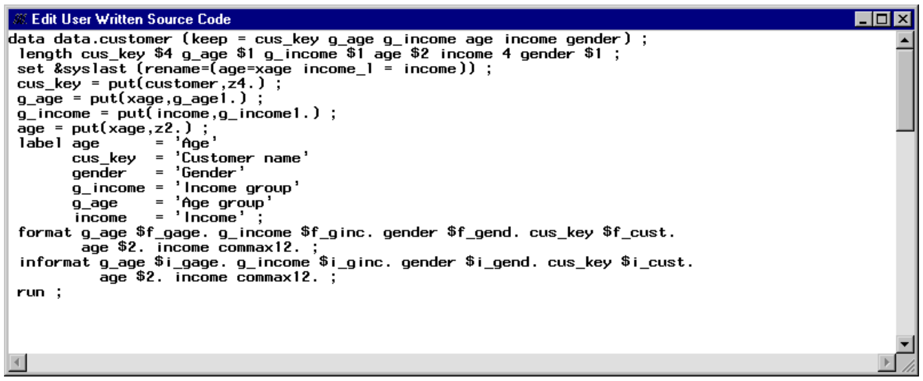
SASWarehouse Administrator Generated

User Written

Source Code Library: Source Code

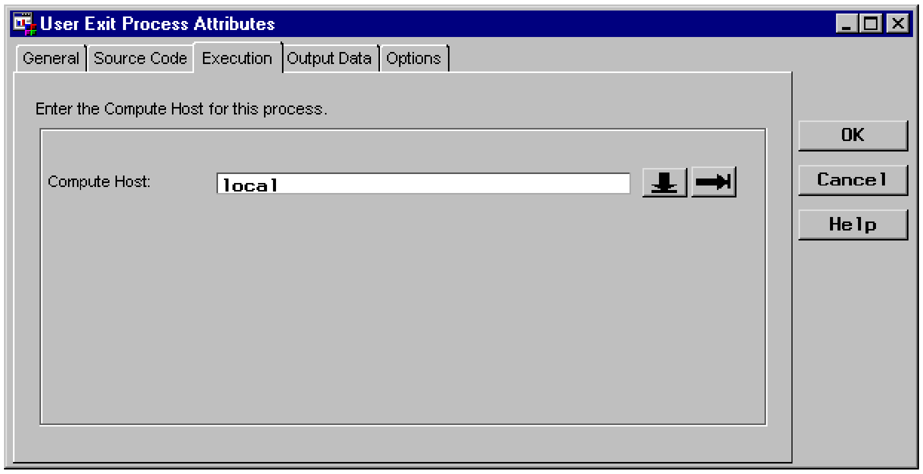
Catalog Entry Name: Source.Customer.Source (catalog.entry.type)

Click [Edit] to open an editing window where you can enter, view, or update the specified code. User-written code can supplement the code that SAS/Warehouse Administrator generates or it can completely replace the code generated by SAS/Warehouse Administrator. Here is an example of code for a User Exit process:



```
data data.customer (keep = cus_key g_age g_income age income gender) ;
 length cus_key $4 g_age $1 g_income $1 age $2 income 4 gender $1 ;
 set &syslast (rename=(age=xage income_l = income)) ;
 cus_key = put(customer,z4.) ;
 g_age = put(xage,g_age1.) ;
 g_income = put( income,g_income1.) ;
 age = put(xage,z2.) ;
 label age      = 'Age'
       cus_key  = 'Customer name'
       gender   = 'Gender'
       g_income = 'Income group'
       g_age    = 'Age group'
        income  = 'Income' ;
 format g_age $f_gage. g_income $f_ginc. gender $f_gend. cus_key $f_cust.
        age $2. income commax12. ;
 informat g_age $i_gage. g_income $i_ginc. gender $i_gend. cus_key $i_cust.
          age $2. income commax12. ;
 run ;
```

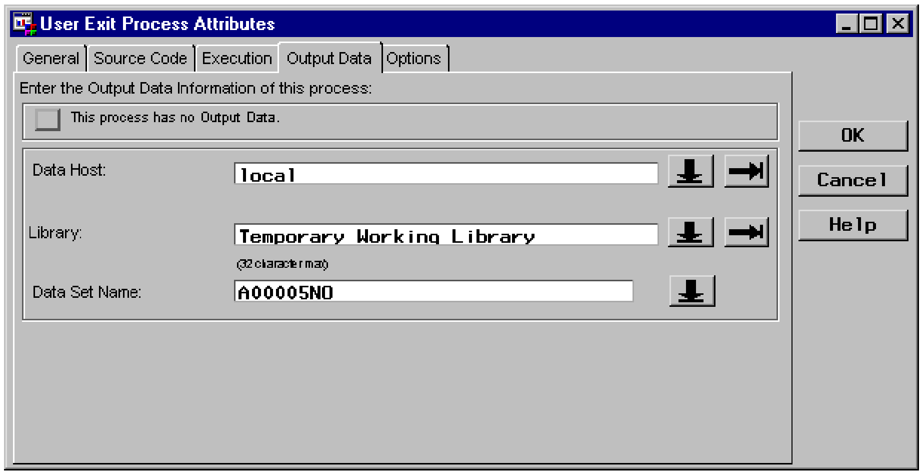
Execution Tab — specifies the host on which you want to run the User Exit process, which for this example is the local host.



Enter the Compute Host for this process.

Compute Host: local

Output Data Tab specifies the name and location of the intermediate work table(s) if output is produced by the User Exit process.



Enter the Output Data Information of this process:

☐ This process has no Output Data.

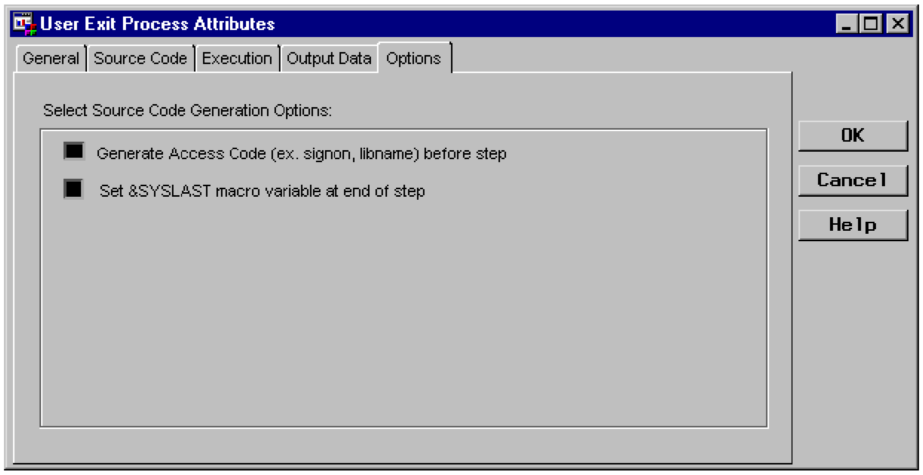
Data Host: local

Library: Temporary Working Library

(32 character max)

Data Set Name: A00005NO

Options Tab specifies options for generating the source code for the process. For this example, both options are selected, which is the default setting.



Select Source Code Generation Options:

■ Generate Access Code (ex. signon, libname) before step

■ Set &SYSLAST macro variable at end of step

The options available on this tab are

- **`Generate Access Code (ex.  signon, libname) before step`** allows you to control whether SAS/Warehouse Administrator generates the code to access the preceding objects, such as sign on code, and the LIBNAME statement. If selected, SAS/Warehouse Administrator generates the code that is required to load or run the current object or process and the code that is required to access the objects or processes that precede the current one. It assumes that code has already been generated or is already in place to load or run the objects or processes that precede the current one.
- **`Set &SYSLAST macro variable at end of step`** allows you to control whether SAS/Warehouse Administrator generates code that sets the &SYSLAST macro variable to the name of the last table that was created or updated.

Example: Defining Data Transfer Process Properties to Move Data from Remote Host to Local Host

Overview

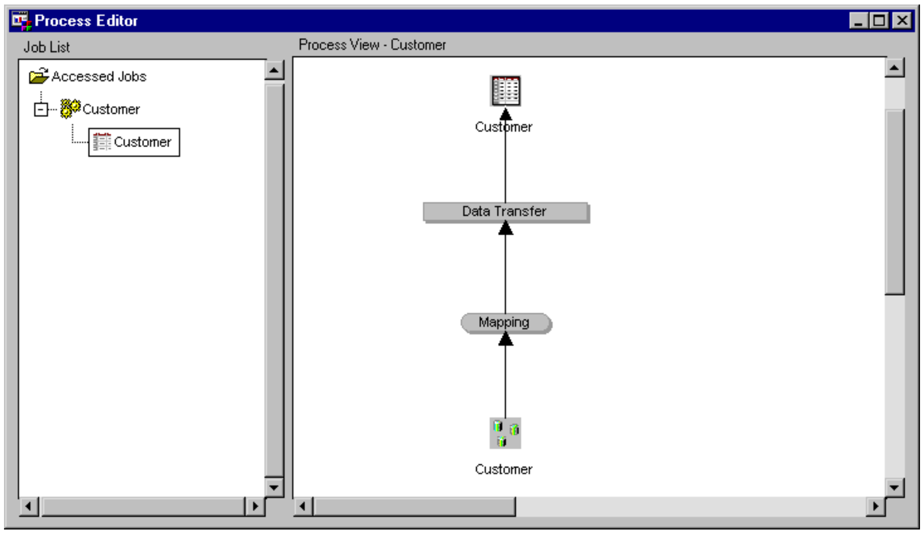
This example defines the Data Transfer process properties to move data from a remote host to the local host. A Data Transfer process is required for the Job because the input source and output table reside on different hosts. The properties specify that SAS/Warehouse Administrator will generate the necessary code to move the data. The Data Transfer process will copy the Customer input data, which is on a remote host, to an intermediate work table on the local host (the host where SAS/Warehouse Administrator is installed). The Mapping process then would map the columns from the intermediate work table to the target table, which is on the local host.

For an example about adding a Data Transfer process, see “Example: Adding a Data Transfer Process to a Process Flow” on page 272.

Note: The following explanation describes the metadata and methods used to achieve the desired results. It is assumed that the appropriate Data Warehouse Environment, Data Warehouse, Subject, data stores, Job, and Process Flow exist. △

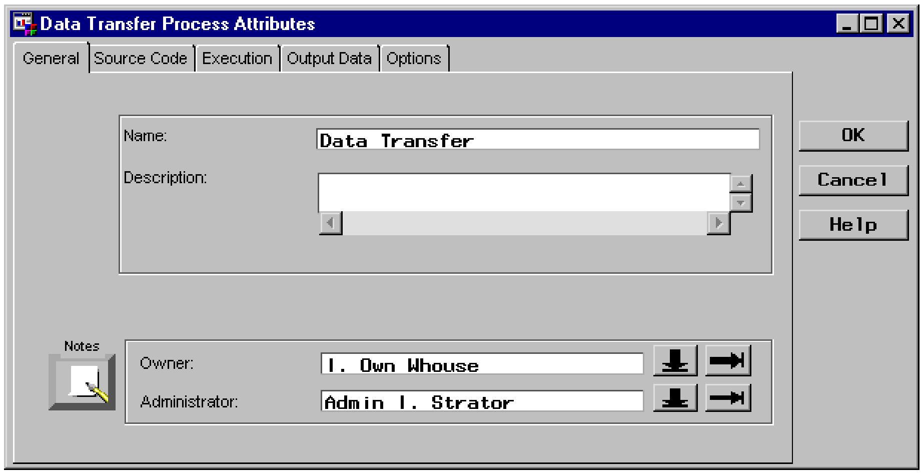
Defining Data Transfer Properties

In a Job Process Flow, a Data Transfer process can be added either below the output table or above the input source as appropriate. For this example, the Data Transfer process was added below the output table as shown in the following Process Flow:



In the Job Process Flow, position the cursor on the Data Transfer process icon, click the right mouse button, and select **`Properties`**. The Data Transfer Process Attributes window displays.

General Tab specifies a name, a description, an owner, and an administrator for the Data Transfer process.


Data Transfer Process Attributes

General | Source Code | Execution | Output Data | Options

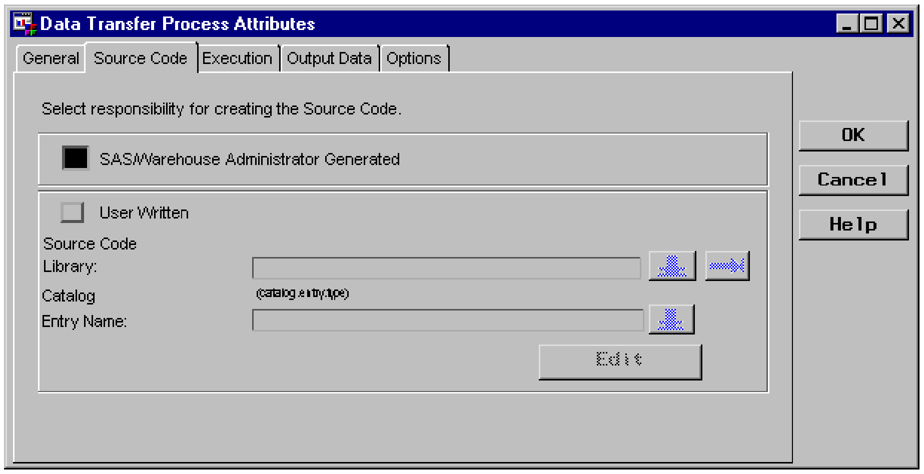
Name: Data Transfer

Description:

Owner: I. Own Whouse

Administrator: Admin I. Strator

Source Code Tab specifies who supplies the source code for the process, which for this example is SAS/Warehouse Administrator.



Select responsibility for creating the Source Code.

■ SAS/Warehouse Administrator Generated

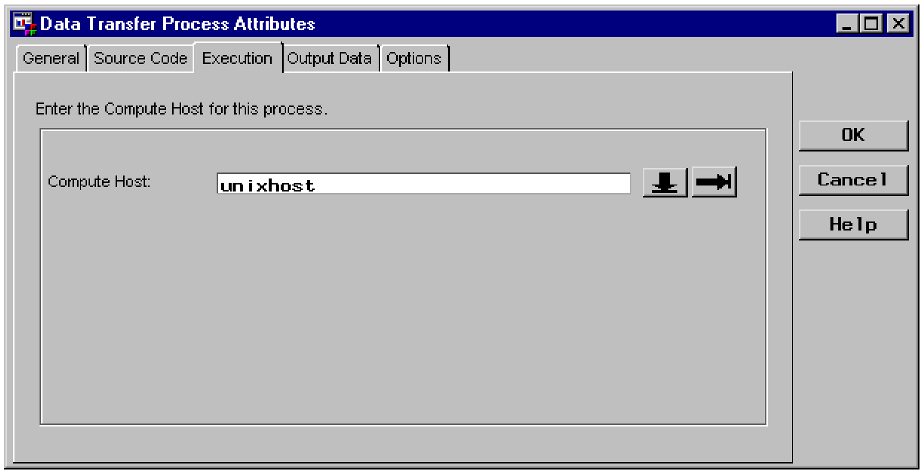
☐ User Written

Source Code Library:

Catalog Entry Name: (catalog.entry.type)

Note: When SAS/Warehouse Administrator generates the code, it uses SAS/CONNECT software and PROC UPLOAD or PROC DOWNLOAD to move the data. △

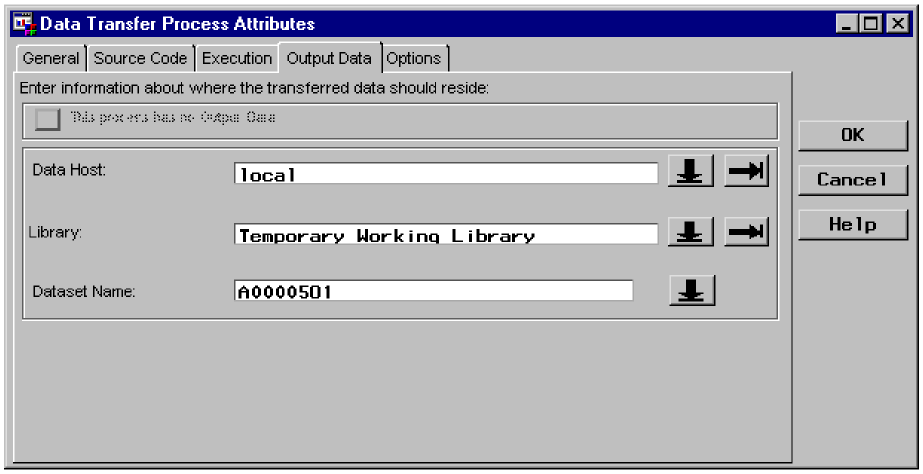
Execution Tab specifies the host on which to run the Data Transfer process, which for a Data Transfer is always the remote host.



Enter the Compute Host for this process.

Compute Host: unixhost

Output Data Tab specifies the name and location of the intermediate work table(s) produced by the Data Transfer process.


Data Transfer Process Attributes

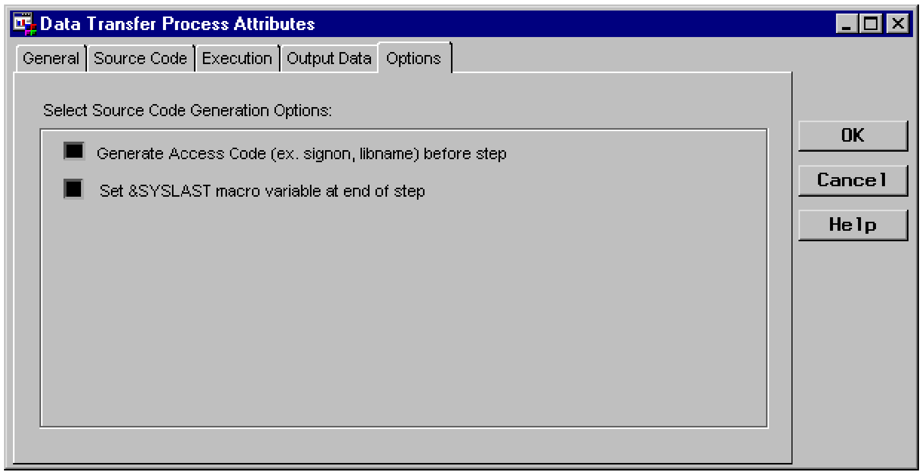
General | Source Code | Execution | Output Data | Options

Enter information about where the transferred data should reside:

☐ This process has no Output Data

Data Host:	local
Library:	Temporary Working Library
Dataset Name:	A0000501

Options Tab specifies options for generating the source code for the process. For this example, both options are selected, which are the default settings.


Data Transfer Process Attributes

General | Source Code | Execution | Output Data | Options

Select Source Code Generation Options:

■ Generate Access Code (ex. signon, libname) before step

■ Set &SYSLAST macro variable at end of step

The options available on this screen are

- **Generate Access Code (ex. signon, libname) before step** allows you to control whether SAS/Warehouse Administrator generates the code to access the preceding objects, such as sign on code, and the LIBNAME statement. If selected, SAS/Warehouse Administrator generates the code that is required to load or run the current object or process and the code that is required to access the objects or processes that precede the current one. It assumes that code has already been generated or is already in place to load or run the objects or processes that precede the current one.

  For example, when SAS/Warehouse Administrator generates code for a Data Transfer process, it generates the statements that are necessary to access the table that precedes the Data Transfer in the Job, but it will not generate any code that might be required to load that table. SAS/Warehouse Administrator assumes that the table has already been loaded.

- **`Set &SYSLAST macro variable at end of step`** allows you to control whether SAS/Warehouse Administrator generates code that sets the &SYSLAST macro variable to the name of the last table that was created or updated.

Note: By default, both of the source code options are selected. If you specify that SAS/Warehouse Administrator generates code for the process (on the Source Code tab), then you should leave these source code options selected. If you are supplying the source code, you might need to select or deselect these options as needed. △

Example: Defining Record Selector Process Properties

Overview

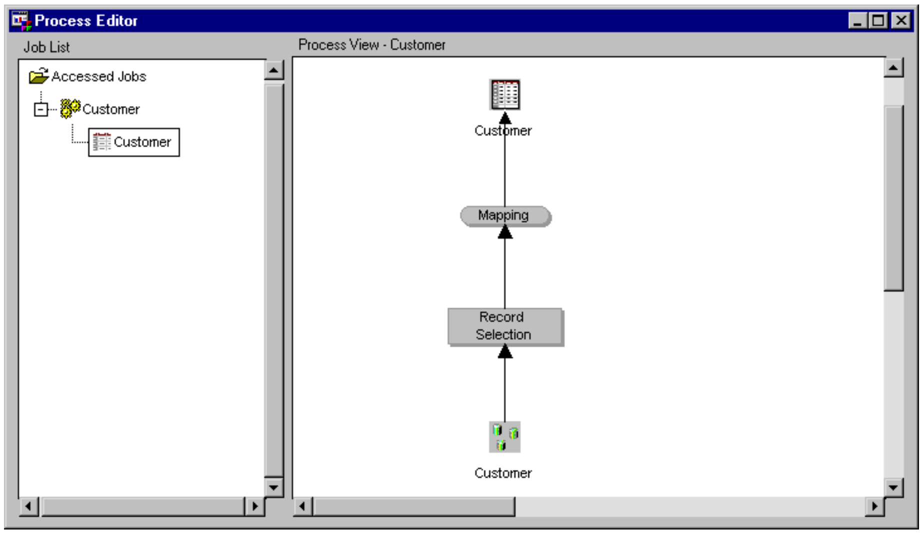
This example defines the Record Selector process properties to subset data from the ODD prior to loading it into the Detail Table output table.

For an example about adding a Record Selector process, see "Example: Adding a Record Selector Process to a Process Flow" on page 274.

Note: The following explanation describes the metadata and methods used to achieve the desired results. It is assumed that the appropriate Data Warehouse Environment, Data Warehouse, Subject, data stores, Job, and Process Flow exist. △

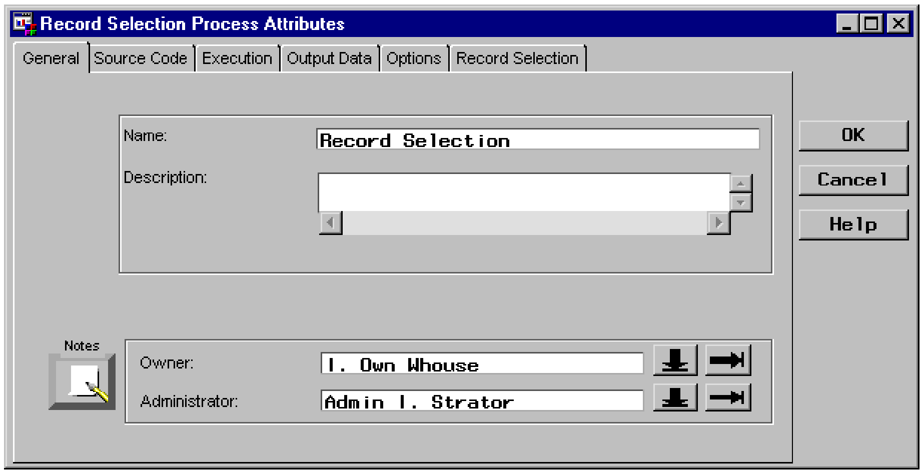
Defining Record Selector Properties

In a Job Process Flow, a Record selector process can be added above the input source as follows:



In the Job Process Flow, position the cursor on the Record Selector process icon, click the right mouse button, and select **`Properties`**. The Record Selection Process Attributes window displays.

General Tab	specifies a name, a description, an owner, and an administrator for the Record Selector process.



Record Selection Process Attributes

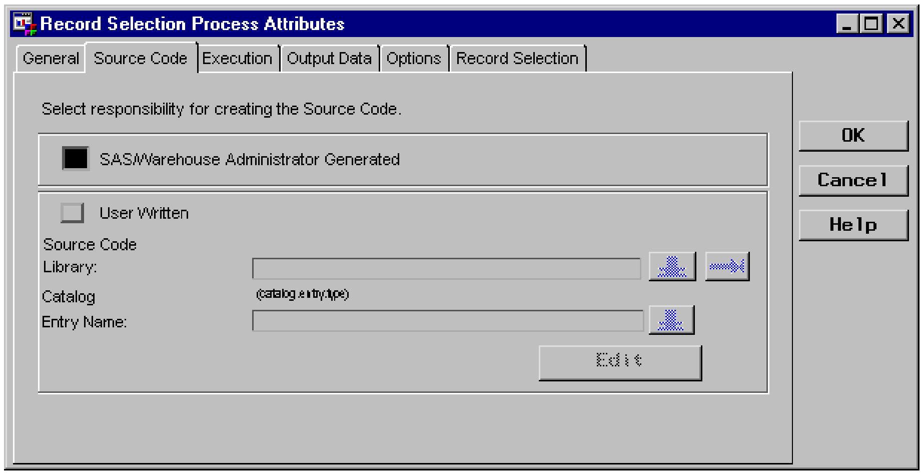
Name: Record Selection

Description:

Owner: I. Own Whouse

Administrator: Admin I. Strator

Source Code Tab specifies who generates the code for the Mapping process, which for this example is SAS/Warehouse Administrator.



Record Selection Process Attributes

Select responsibility for creating the Source Code.

■ SAS/Warehouse Administrator Generated

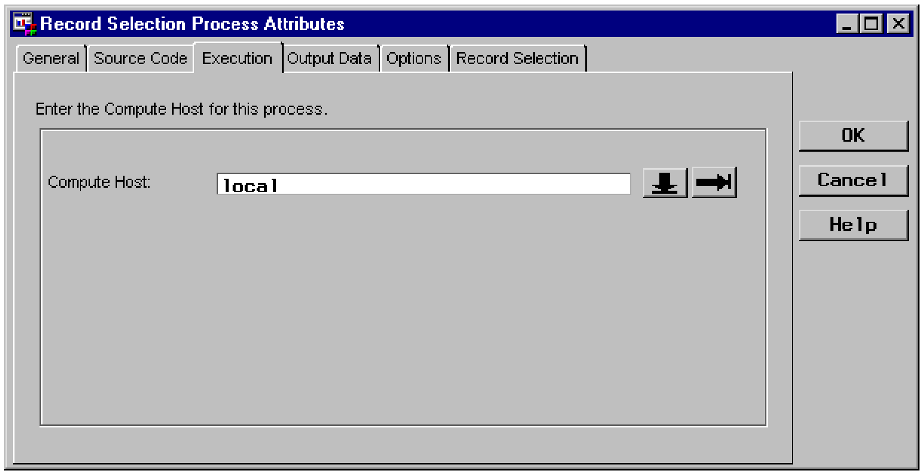
□ User Written

Source Code Library:

Catalog (catalog.entry.type)

Entry Name:

Execution Tab specifies the host on which you want to run the Record Selector process, which for this example is the local host.

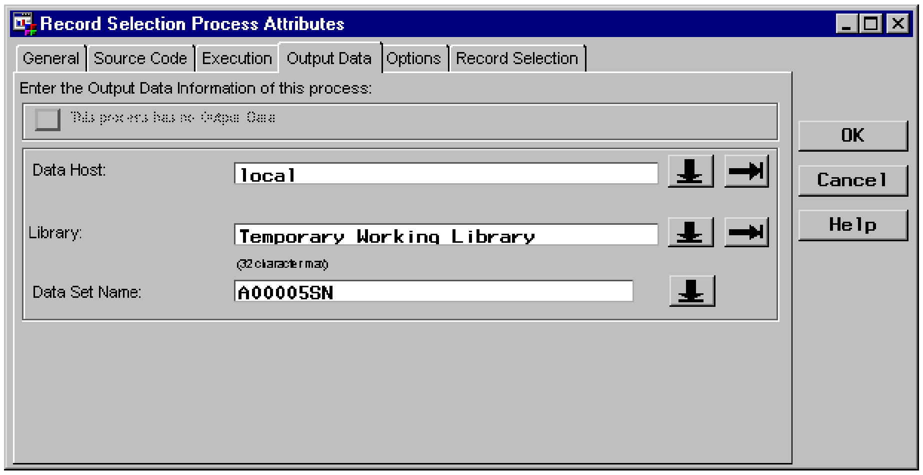


Record Selection Process Attributes

Enter the Compute Host for this process.

Compute Host: local

Output Data Tab
: specifies the name and location of the intermediate work table(s) produced by the Record Selector process.



Enter the Output Data Information of this process:

This process has no Output Data

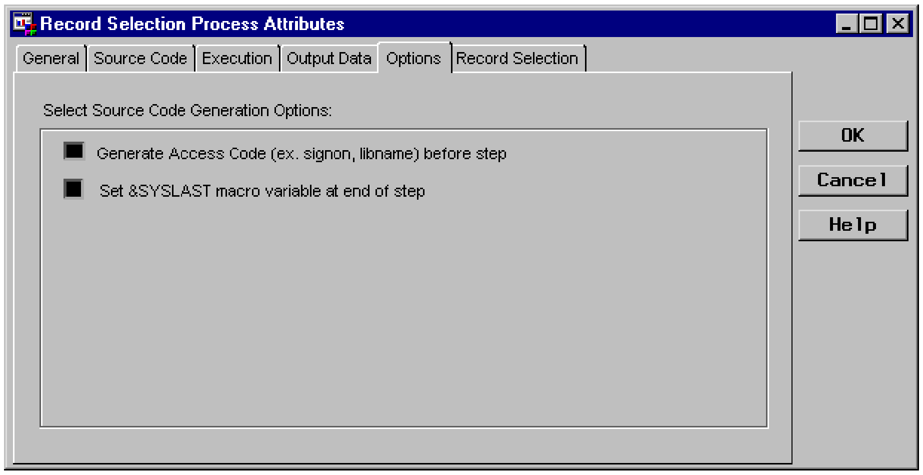
Data Host: local

Library: Temporary Working Library

(32 character max)

Data Set Name: A00005SN

Options Tab
: specifies options for generating the source code for the process. For this example, both options are selected, which is the default setting.



Select Source Code Generation Options:

■ Generate Access Code (ex. signon, libname) before step

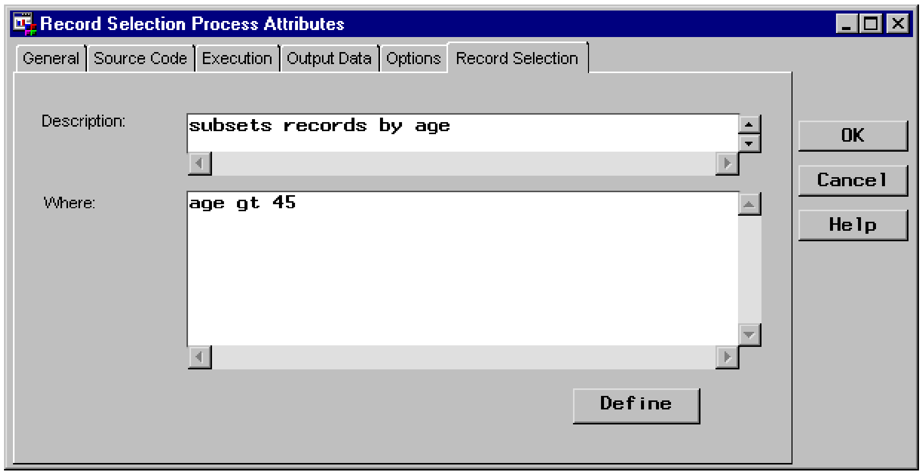
■ Set &SYSLAST macro variable at end of step

The available options on this tab are

- **`Generate Access Code (ex. signon, libname) before step`** allows you to control whether SAS/Warehouse Administrator generates the code to access the preceding objects, such as sign on code, and the LIBNAME statement. If selected, SAS/Warehouse Administrator generates the code that is required to load or run the current object or process and the code that is required to access the objects or processes that precede the current one. It assumes that code has already been generated or is already in place to load or run the objects or processes that precede the current one.
- **`Set &SYSLAST macro variable at end of step`** allows you to control whether SAS/Warehouse Administrator generates code that sets the &SYSLAST macro variable to the name of the last table that was created or updated.

Note: By default, both of the source code options are selected. If you specify that SAS/Warehouse Administrator generates code for the process (on the Source Code tab), then you should leave these source code options selected. If you are supplying the source code, you might need to select or deselect these options as needed. △

Record Selection Tab
: specifies a condition that the data must satisfy before SAS reads it. SAS/Warehouse Administrator will generate an SQL WHERE clause from the information you enter. You can enter the condition directly or click Define to open the Expression Builder window, in which you can specify the condition. For example, you could enter the following condition:



Description: subsets records by age

Where: age gt 45

Example: Editing Load Process Properties to Supply User-Written Code

Overview

With its default settings, you might not need to edit the Load process. However, for example, if you intend to supply user-written code for the Load process and the default is so that the code is generated by SAS/Warehouse Administrator, you must edit the Load process properties. Also, SAS/Warehouse Administrator cannot generate source code for an ODD or a Detail Logical Table Load process. Therefore, for those data stores, you must edit the Load process to supply the user-written code.

For an example of a resulting Load process, see "Editing the Load Process" on page 260.

Writing Code for a Load Process

Load process code creates a table or view. SAS provides a number of tools that you can use to write Load process code:

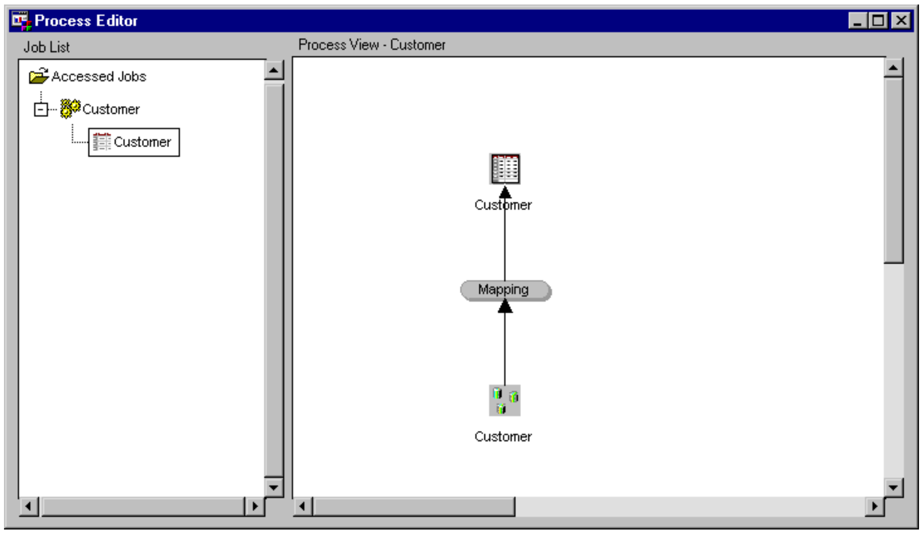
- DATA step
- SQL procedure
- Query window
- External File Interface (EFI)
- SAS/ASSIST software

- □ SAS/TOOLKIT software

For more information about writing Load process code, click Help on the data store's Load process attributes window. Then, select the link for the **`Source Code tab`**, then select **`User-Written Source Code`**.

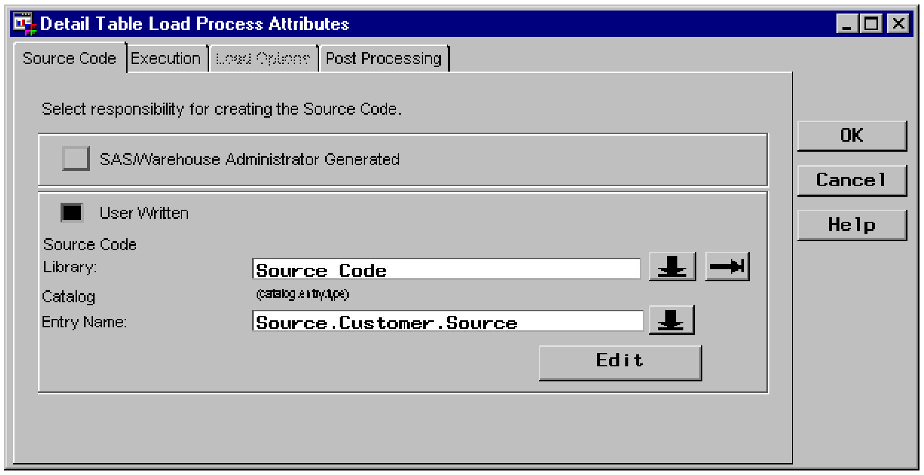
Editing Load Step Properties

The Load process is included automatically in the Process Flow. However, the Load process does not have its own icon; it is associated with the output table.



In the Job Process Flow, position the cursor on the output table, click the right mouse button, and select **`Edit Load Step`**. The Load Process attributes window for the data store displays.

Source Code Tab specifies who supplies the source code for the Job, which for this example is user-written code.



Select responsibility for creating the Source Code.

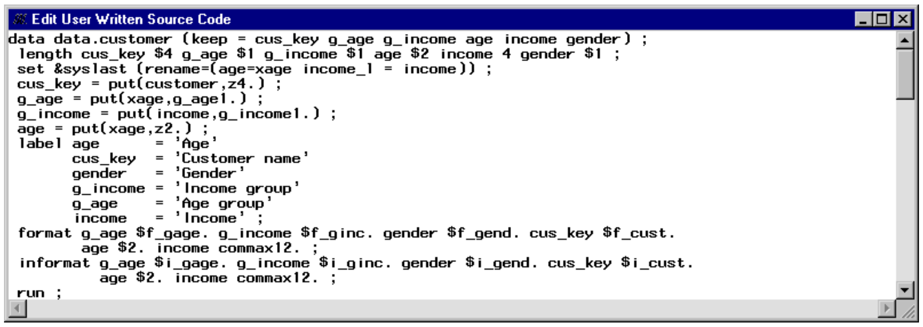
SAS/Warehouse Administrator Generated

User Written

Source Code Library: Source Code

Catalog Entry Name: Source.Customer.Source (catalog.entry.type)

Click Edit to open an editing window in which you can enter, view, or update the specified code. An example of Load process code follows:

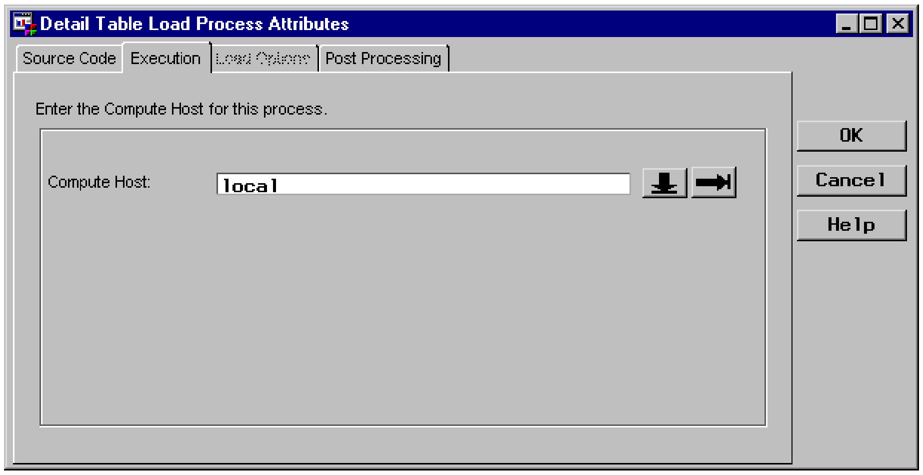


Edit User Written Source Code

```
data data.customer (keep = cus_key g_age g_income age income gender) ;
 length cus_key $4 g_age $1 g_income $1 age $2 income 4 gender $1 ;
 set &syslast (rename=(age=xage income_l = income)) ;
 cus_key = put(customer,z4.) ;
 g_age = put(xage,g_age1.) ;
 g_income = put(income,g_income1.) ;
 age = put(xage,z2.) ;
 label age      = 'Age'
       cus_key  = 'Customer name'
       gender   = 'Gender'
       g_income = 'Income group'
       g_age    = 'Age group'
       income   = 'Income' ;
 format g_age $f_gage. g_income $f_ginc. gender $f_gend. cus_key $f_cust.
        age $2. income commax12. ;
 informat g_age $i_gage. g_income $i_ginc. gender $i_gend. cus_key $i_cust.
          age $2. income commax12. ;
 run ;
```

Note: For the output table, its properties window Columns tab must match the columns to be loaded. If not, the Mapping process will not match the columns loaded into the data store. △

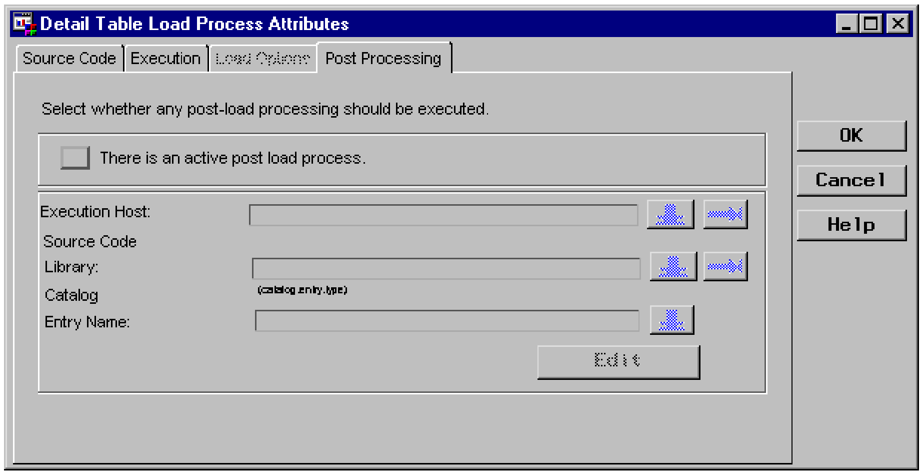
Execution Tab specifies the host on which you want to run the Load process. This example specifies the local host.



Enter the Compute Host for this process.

Compute Host: local

Post Processing Tab specifies code to be executed after the data store is loaded. For example, you might want to register code that sends an e-mail indicating that the Load process has finished.



Select whether any post-load processing should be executed.

☐ There is an active post load process.

Execution Host:

Source Code Library:

Catalog Entry Name: (catalog.entry.type)

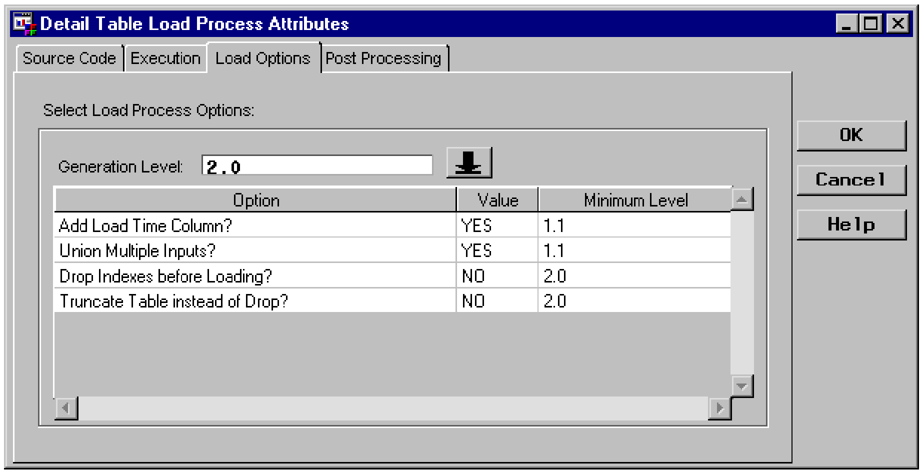
Example: Editing Load Process Properties for SAS/Warehouse Administrator Generated Code

Overview

If you want SAS/Warehouse Administrator to generate the code for the Load process, you do not need to edit the Load process. However, one property that you might need to edit is the Load Options tab, which is the only tab discussed in this example.

Editing Load Step Properties

Load Options Tab	specifies the level of code that SAS/Warehouse Administrator will use to generate the code and it also lists all possible Load process options.



Select Load Process Options:

Generation Level: 2.0

Option	Value	Minimum Level
Add Load Time Column?	YES	1.1
Union Multiple Inputs?	YES	1.1
Drop Indexes before Loading?	NO	2.0
Truncate Table instead of Drop?	NO	2.0

Code generation level **`2.0`** is the level of code generated by SAS/Warehouse Administrator Release 2.0 (the current release). It uses SAS/ACCESS LIBNAME statements to read and write data in DBMS format. (SQL Pass-Through views are used for some tasks associated with DBMS tables, such as reading or writing DBMS indexes.)

Code generation level **`1.1`** is the level of code generated by SAS/Warehouse Administrator Releases 1.1, 1.2, and 1.3. It creates an SQL Pass-Through view to access DBMS data and uses PROC DBLOAD to load data into DBMS tables.

For more information about the levels of code, when to use one or the other, and other Load process options such as the **`Load Time`** column, click Help on a specific data store's Load Process Attributes window.

CHAPTER 15 Scheduling Jobs

Overview

This chapter describes how to schedule, track, and view Jobs in SAS/Warehouse Administrator. The CRON and AT scheduling servers are explicitly supported. It is possible to use a Null Scheduler definition to generate a Job that can be used with other scheduling servers.

Preparing for Job Scheduling

Before you can schedule a Job in SAS/Warehouse Administrator, you must do some preparation, as follows:

Host configuration	□ For a production data warehouse in which Jobs will be scheduled and tracked through SAS/Warehouse Administrator, we strongly recommend that all Jobs Information libraries be placed under the control of a SAS/SHARE server. For details, see "Jobs Information Libraries and SAS/SHARE Software" on page 24.
Global metadata	□ Create the SAS library that you will register as the Jobs Information library for a given Warehouse Environment or Data Warehouse, as described in "Example: Creating a Jobs Information Library" on page 83. □ Register the Jobs Information library for a given Warehouse Environment or Data Warehouse, as described in "Registering Jobs Information Libraries" on page 306. □ Create a host definition for the host on which the scheduling server application (such as CRON) runs. For details about host definitions, see "Host Definitions" on page 86.

- □ Create the appropriate scheduling server definition, as described in “Scheduling Server Definitions” on page 99.

Registering Jobs Information Libraries

A Jobs Information library is a SAS library that contains status information for Jobs that have been scheduled through the Job Properties window in SAS/Warehouse Administrator. If Job tracking is enabled for a given Job, when the Job executes, it will update its status in the appropriate Jobs Information library. The Job Viewer window reads the Jobs Information library to display information about Jobs that have been submitted.

After you have created an appropriate library definition, you can make that library the Jobs Information library for a given Environment or Data Warehouse. A Warehouse Environment and its Data Warehouses can share one Jobs Information library, or they could have separate Jobs Information libraries.

Registering a Jobs Information Library

Here is the preferred way to register a SAS library as the Jobs Information library for a given Warehouse Environment or Data Warehouse.

1 If you have not done so already, create an appropriate library definition, as described in “Example: Creating a Jobs Information Library” on page 83.

2 Display the Job Hierarchy view in the Process Editor.

   If you are currently in the SAS/Warehouse Administrator Explorer, from the main menu, select

   Tools ► Process Editor

   If you are currently in the Job List view of the Process Editor, right-click the background of the Job List view, and select

   View ► Job Hierarchy

3 In the Job Hierarchy view, left-click the Warehouse Environment or Data Warehouse that needs a Jobs Information library.

   In the Process Editor, a rectangle appears around the selected Environment or Data Warehouse.

4 Right-click the Warehouse Environment or Data Warehouse and select

   Job Info Library ► Select Existing Library

   A list of libraries defined in the current Environment displays.

5 Select the library you defined in Step 1.

   The library defined in Step 1 is now the Jobs Information library for the Warehouse Environment or Data Warehouse that you selected.

Scheduling Jobs

Before SAS/Warehouse Administrator can generate the code that schedules a Job, at a minimum, you must have

- □ a scheduling server definition for the computer where the Job will run

- □ an appropriate Jobs Information library.

For details about the preparation required for Job scheduling, see "Preparing for Job Scheduling" on page 305.

To have SAS/Warehouse Administrator schedule a Job, at a minimum, you must open the properties window for the Job and enter information on the Date/Time tab and the Server tab. For details about these tabs, click Help on the Job Properties window.

Tracking Jobs

If Job tracking is enabled for a given Job, when the Job executes, it will update its status in the appropriate Jobs Information library. The Job Viewer window reads the Jobs Information library to display information about Jobs that have been submitted.

Job tracking through SAS/Warehouse Administrator is enabled by default. A Job scheduled through SAS/Warehouse Administrator will be tracked through SAS/ Warehouse Administrator unless job tracking has been disabled for the Job on the Prolog/Epilog tab of the Job Properties window.

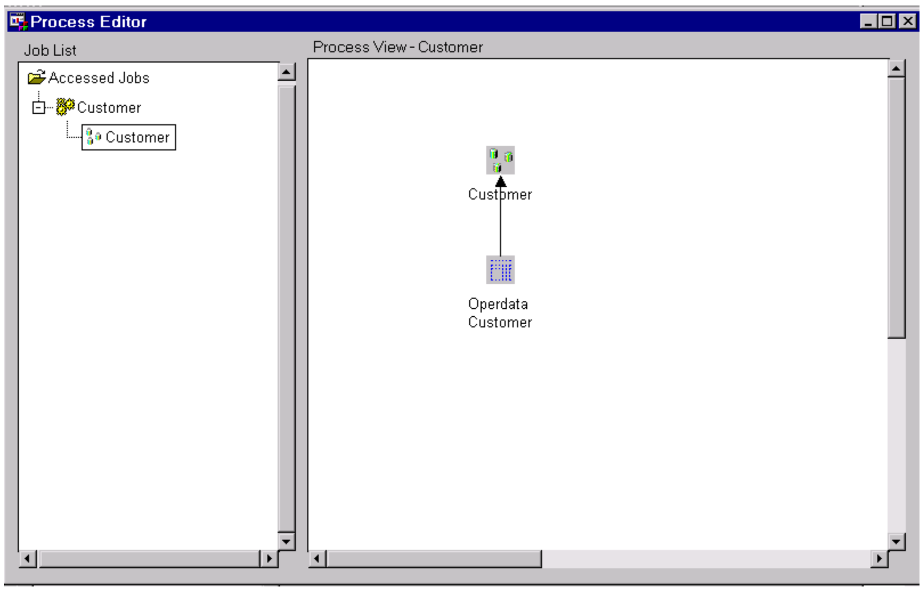
Example: Scheduling and Tracking a Job with the AT Command

This example describes how to schedule and track a Job for a local data store on a Microsoft Windows or Windows NT computer. The appropriate scheduling server definition and Jobs Information library are assumed to exist, although brief instructions for adding these items are included in the example.

Note: The steps for scheduling a Job using System V CRON are very similar to the steps described here. The main difference is that you would select a CRON definition on the Server tab for the Job. △

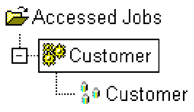
1 Display the Job for the data store. In the SAS/Warehouse Administrator Explorer, right-click the data store and select `Process`.

The Job will be displayed in the Process Editor, as follows:

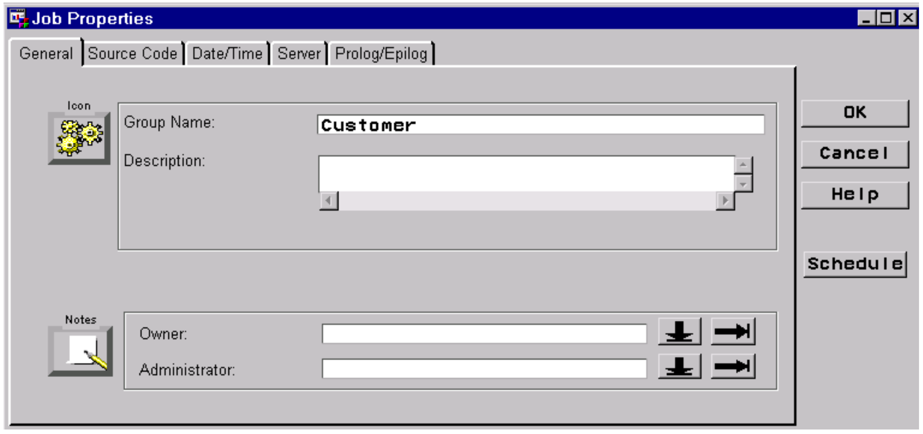


In the left panel of the Process Editor, the data store's Job and output tables will be listed. In the previous display, the output table for the Job has a rectangle around it. Note that the output table has a parent. This parent is the Job.

2 (Optional) Position the cursor on the Job, and click the left mouse button to select it, as follows:

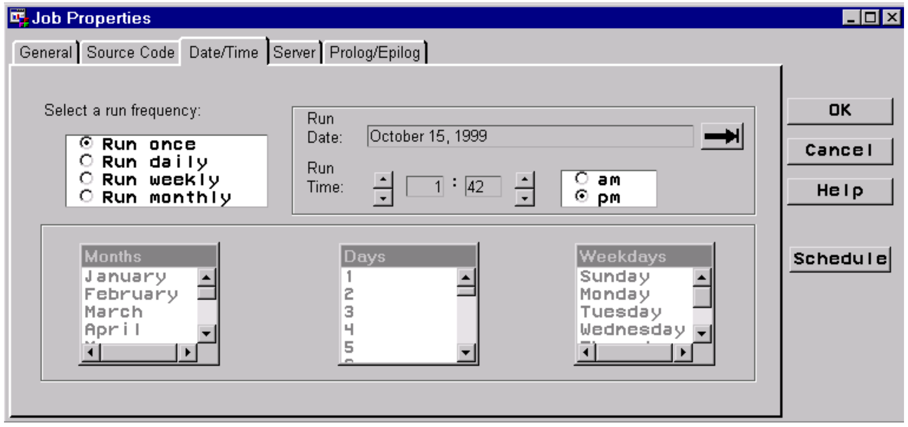


3 Position the cursor on the Job, click the right mouse button, and select **`Properties`**. The Job Properties window displays, as follows:



Group Name: Customer
Description:
Owner:
Administrator:

4 Select the Date/Time tab. The tab displays, as follows:

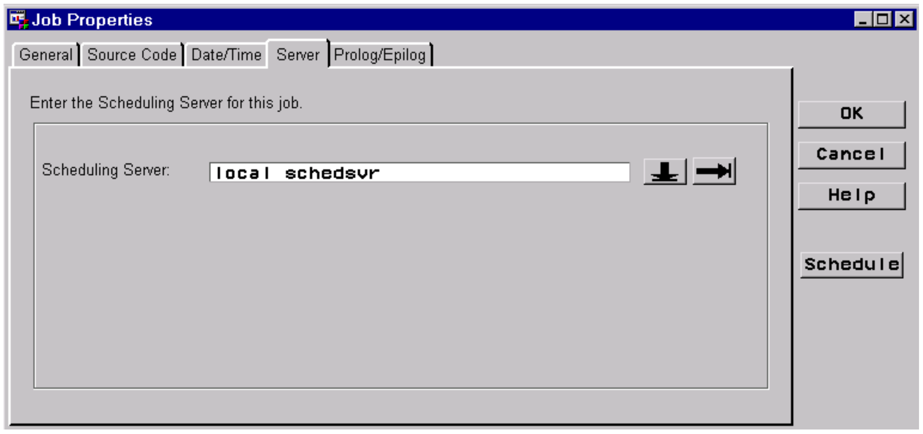


Select a run frequency:
[✓ Run once]
Run daily
Run weekly
Run monthly

Run Date: October 15, 1999
Run Time: 1 : 42 am [✓ pm]

At a minimum, use the arrows to select a date and a time for the Job to execute. For details about the fields on this tab, click Help.

5 Select the Server tab. The tab displays, as follows:

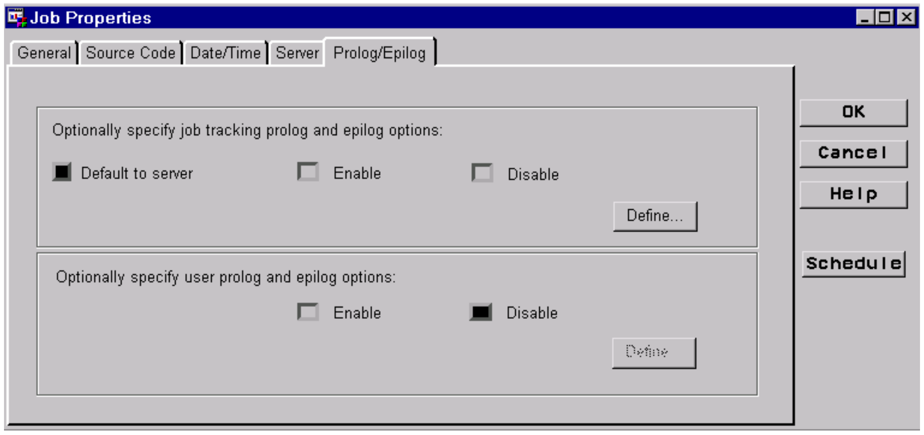


Enter the Scheduling Server for this job.

Scheduling Server: local schedsvr

Use the arrow to select a scheduling server definition that is appropriate for the Job. In the current example, the definition should generate an AT command that will execute the Job on the local host. To see a scheduling server definition that would be appropriate for the current example, see "Example: Creating an AT Scheduling Server Definition" on page 100.

6 Select the Prolog/Epilog tab. The tab displays, as follows:



Optionally specify job tracking prolog and epilog options:

[✓ Default to server] Enable Disable

Optionally specify user prolog and epilog options:

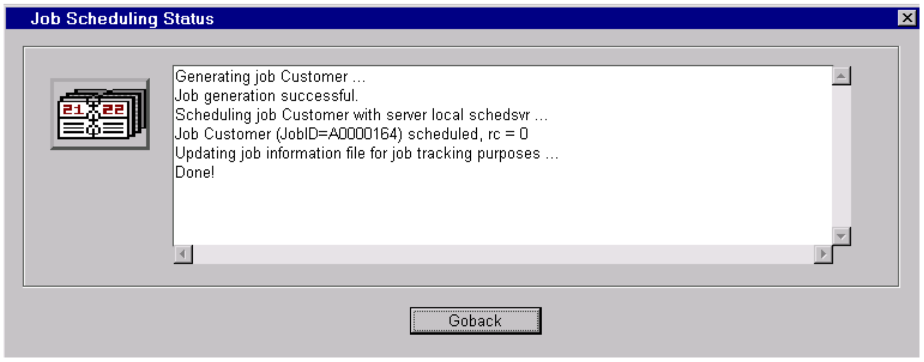
Enable [✓ Disable]

In the Prolog/Epilog tab above, note that the **`Default to server`** option is selected. This means that the Tracking tab of the properties window for the scheduling server definition specified for this Job will determine whether this Job will be tracked in SAS/Warehouse Administrator. For the current example, assume that the definition specified on the Server tab has its Job tracking option selected.

You are now finished entering the metadata required to schedule this Job.

7 Click Schedule.

8 Click Yes when asked if you want to save the metadata for the current Job. The Job Scheduling Status window displays. It presents a series of messages about the status of the Job, as follows:



```
Generating job Customer ...
Job generation successful.
Scheduling job Customer with server local schedsvr ...
Job Customer (JobID=A0000164) scheduled, rc = 0
Updating job information file for job tracking purposes ...
Done!
```

When the messages are all displayed, click Go Back to return to the Job Properties window.

Note: If you get a message that says that the maximum length for an AT command has been exceeded, you must edit the scheduling server definition that is specified for this Job. Click Go Back to return to the Job Properties window. Click the Server tab, then click the right arrow and select **`Edit existing scheduling server`**. For details about what updates are required to fix the problem, see "Preparing to Create an AT Scheduling Server Definition" on page 100. △

Note: If you get a message about the Jobs Information library not being defined, you must create a new one—or register an existing Jobs Information library—before you can schedule a Job in the current Warehouse Environment or Data Warehouse. △

It is possible that an appropriate library has been defined, but has not yet been registered as the Jobs Information library for the current Warehouse Environment or Data Warehouse. In that case, cancel the scheduled Job and register the Jobs Information library as described in "Registering a Jobs Information Library" on page 306. Having done that, you can come back to the same Job and schedule it again. These steps will prevent you from creating multiple Jobs Information libraries.

Alternatively, you might know that an appropriate Jobs Information library does not exist. In that case, click Yes when asked if you want to create a Jobs Information library. A properties window displays. For details about that window, click Help.

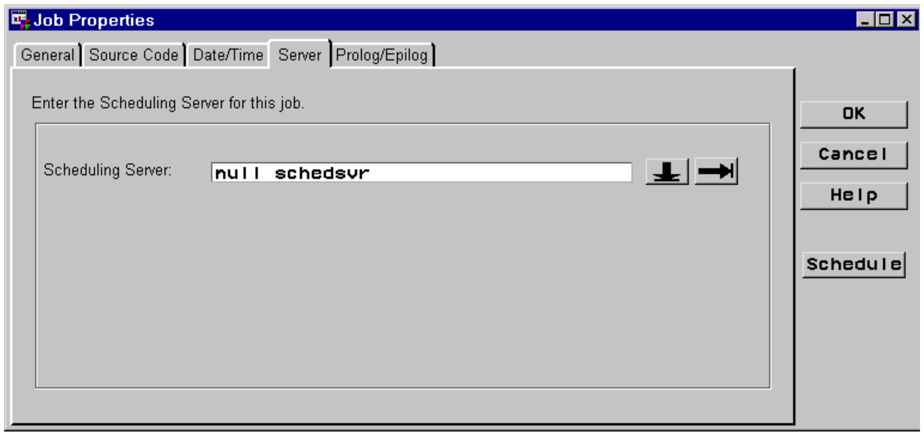
9 From the Job Properties window, click OK. You are returned to the Process Editor.

After you have scheduled a Job, you can view it, as described in "Viewing Scheduled Jobs" on page 311.

Summary: Scheduling a Job with the Null Scheduler

This example summarizes how to schedule a Job using the Null Scheduler definition rather than an AT or CRON definition. For a discussion of how this can be useful, see "Summary: Creating and Using a Null Scheduling Server Definition" on page 105.

The steps required to perform this task are very similar to those described in "Example: Scheduling and Tracking a Job with the AT Command" on page 307. The critical difference is that on the Server tab of the Job Properties window, you would specify a Null Scheduler definition. An example of such a Server tab follows:



Enter the Scheduling Server for this job.

Scheduling Server: null schedsvr

Viewing Scheduled Jobs

Here is one way to view all scheduled Jobs for a given Warehouse Environment or Data Warehouse.

1 Display the Job Hierarchy view in the Process Editor.

If you are currently in the SAS/Warehouse Administrator Explorer, from the main menu, select

Tools ▶ Process Editor

If you are currently in the Job List view of the Process Editor, right-click the background of the Job List view, and select

View ▶ Job Hierarchy

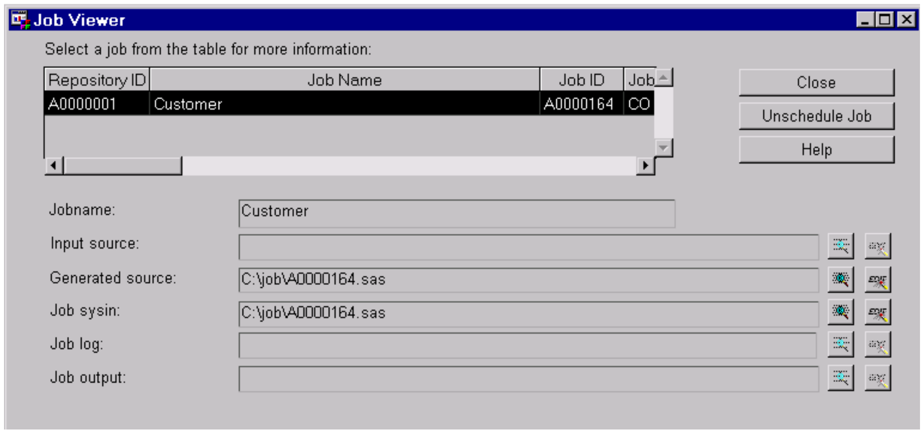
2 In the Job Hierarchy view, select the Warehouse Environment or Data Warehouse whose Jobs you want to view.

To view the scheduled Jobs for data stores at the Environment level (any data store within an ODD Group in the Explorer), select the Environment. To view the scheduled Jobs for data stores in a Data Warehouse, select that Data Warehouse. It is possible that an Environment and a Data Warehouse could share the same Jobs Information library.

In the Process Editor, a rectangle appears around the selected Environment or Data Warehouse.

3 From the main menu, select

Tools ▶ Job Status Viewer



Select a job from the table for more information:

Repository ID	Job Name	Job ID	Job
A0000001	Customer	A0000164	CO

Jobname: Customer

Input source:

Generated source: C:\job\A0000164.sas

Job sysin: C:\job\A0000164.sas

Job log:

Job output:

For details about Job Viewer window, click Help.

CHAPTER

16 Exploiting Warehouse Metadata

Overview: Metadata Repositories

When you open a Warehouse Environment and a Data Warehouse in the SAS/Warehouse Administrator Explorer, two metadata repositories are active:

libref _MASTER	specifies the current Environment repository, which stores metadata for any data stores defined at the Environment level, along with host definitions and other global metadata. _MASTER contains references to any Data Warehouse repositories (libref _DWMD).
libref _DWMD	specifies the current Warehouse repository, which stores metadata for any data stores defined at the Data Warehouse level.

Note: Only two metadata repositories are active at any given time: _MASTER and _DWMD. △

When you open a Warehouse Environment in the Explorer, a libref to the corresponding _MASTER library is assigned. You cannot open a second Environment from the Explorer. When you exit the Explorer, the libref to _MASTER is unassigned.

When you open a Data Warehouse in the Explorer, a libref to the corresponding _DWMD library is assigned. If you open a second Data Warehouse, the libref to the first one is unassigned, and the libref to the second one is assigned.

Maintaining a metadata repository is a matter of maintaining the corresponding Warehouse Environment or Data Warehouse, as described previously in this document. Be sure to back up the metadata repositories for an Environment and its Data Warehouses.

Impact of Metadata Repositories on User Operations

Some user operations that involve two objects in the Explorer or the Process Editor can only be completed if the metadata for these objects is in the same metadata repository. These operations are only permitted if

- the metadata for both objects is defined in the same Data Warehouse
- the metadata for both objects is defined at the Warehouse Environment level
- the metadata for both objects is defined in the same Environment.

For example, cut and paste operations in the SAS/Warehouse Administrator Explorer and Process Editor are restricted to the same metadata repository. This means

- you cannot cut and paste any object between a Data Warehouse and an ODD Group
- you cannot cut an object from one Data Warehouse and paste it into another Data Warehouse
- you can cut an object from one ODD Group and paste it into another ODD Group.

For example, all output tables that you add to a Job must be in the same metadata repository as the Job. Keep this in mind when you add multiple output tables to a Job, as described in “Example: Defining a Job with Multiple Output Tables and Input Sources in a Process Flow” on page 265.

Metadata Details View in the Explorer

You can browse Warehouse metadata through the Metadata Details view in the Explorer. Open the Metadata Details view using one of the following methods:

- Open an Environment in the Explorer.
- From the menu bar, select `View`, then `Metadata Details`.
- Expand the contents of a Data Warehouse.
- Position the cursor on a group or data store and click the left mouse button.

  The metadata tabs for the selected object will be displayed.

The Metadata Details view in the Explorer displays much of the information that you enter when you define warehouse elements. Other information is generated by SAS/Warehouse Administrator, such as the dates and times that the metadata for a warehouse element were created. Some of the tabs—such as the tabs for Notes, Support, and Process—appear only if you have defined metadata for those tabs.

Note: It is recommended that you set the Explorer view to Metadata General when there is no need to search metadata. △

The Metadata Details view can impact performance because it searches and displays the metadata for the specific object.

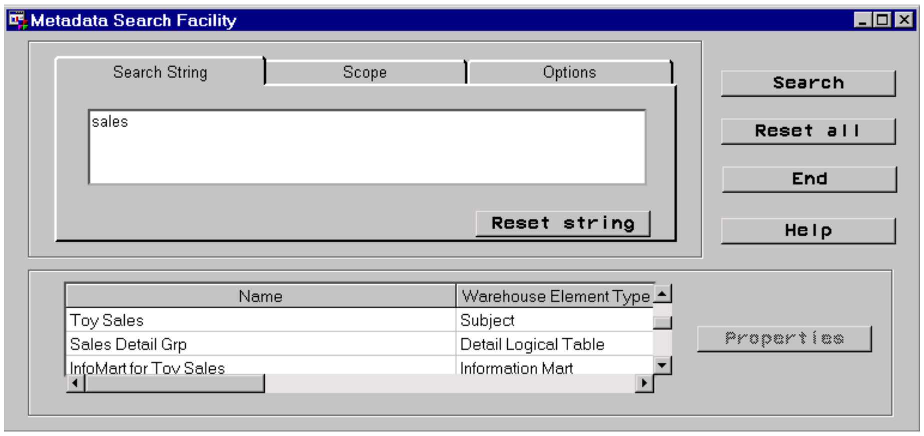
Metadata Search Facility

Use the Metadata Search Facility to search for character strings in the metadata for groups and data stores. To display the Metadata Search Facility:

1. Open an Environment in the Explorer.
2. From the menu bar, select

Tools ▶ Search Metadata

The Metadata Search Facility window displays, as shown next:



sales

Name	Warehouse Element Type
Toy Sales	Subject
Sales Detail Grp	Detail Logical Table
InfoMart for Toy Sales	Information Mart

For details about this window, click Help.

MetaSpace Explorer

Note: This feature is available in SAS/Warehouse Administrator Releases 2.0, 2.1 and 2.2 only. △

The MetaSpace Explorer is a Java applet that enables you to browse metadata that has been exported from SAS/Warehouse Administrator. End users can use their Web browsers to locate tables, charts, graphs, and documentation associated with the Warehouse. For example, business analysts could browse through Warehouse subject areas, focus in on one particular area of interest, request access to that information, and surface detailed data to their desktops or download it for further manipulation.

To export SAS/Warehouse Administrator metadata for the MetaSpace Explorer, follow the instructions in “Exporting Metadata for Groups and Data Stores” on page 316. From the Export Format Selection window, select **`MetaSpace Explorer`**.

The MetaSpace Explorer can be downloaded from the SAS Institute Web site.

1. Go to the SAS Institute home page: **`http://www.sas.com`**.
2. Under *Service & Support*, select *Downloads*.
3. On the *All Downloads* page, select *SAS IntrNet Software*.
4. From the *SAS IntrNet Software* page, under the heading *For Versions 8 & 6 of the SAS System*, select *Java Tools*.
5. From the *Java Tools* page, select the appropriate MetaSpace Explorer.

Exporting Metadata

You can export SAS/Warehouse Administrator metadata so that it can be exploited by other applications such as base SAS, SAS/EIS, and the MetaSpace Explorer.

Exporting Host Definitions and Other Shared Metadata

To export the metadata defined for SAS libraries, hosts, DBMS connections, and contacts:

1 Open an Environment in the Explorer.

2 From the menu bar, select

File ► Setup

The Define Items Used Globally Window displays.

3 In the Define Items Used Globally Window, select the kind of metadata you would like to export (SAS libraries, hosts, etc.).

4 From the menu bar, select

Tools ► Export Metadata

The Export Format Selection window displays. The only format available is **`SAS Datasets`**.

5 From the Export Format Selection window, click OK.

The Export Metadata window displays.

6 Select the library where you would like to save the exported metadata and click OK.

Exporting Metadata for Groups and Data Stores

To export the metadata defined for groups and data stores:

1 Open an Environment in the Explorer.

2 Position the cursor on the group or data store whose metadata you want to export and click the right mouse button.

3 From the pop-up menu, select **`Export Metadata`**.

The Export Format Selection window displays. For details about this window and any subsequent windows, click Help.

Example: Exporting Metadata to SAS/EIS Software

Overview

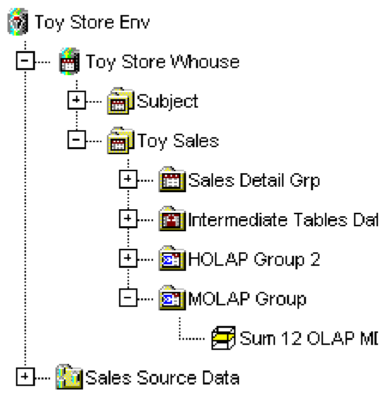
This example illustrates how to export SAS/Warehouse Administrator metadata to SAS/EIS software. The example exports metadata for an OLAP MDDB, which automatically registers the file in the SAS/EIS repository.

You can export metadata for SAS/Warehouse Administrator groups and data stores. For example, you can export the metadata for a Data Table used as a lookup table, you can export an OLAP data store to use in an OLAP application, and you can export an OLAP Group of type HOLAP.

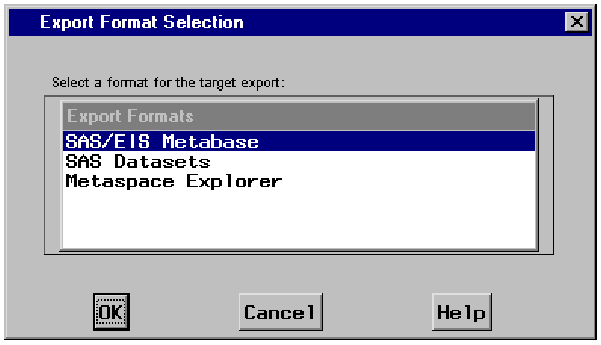
Note: The following explanations assume that the OLAP Group and OLAP MDDB exist. For instructions about creating the OLAP objects, see “Example: Creating Summary Data for a MOLAP Application” on page 208. The example also assumes that SAS/EIS has been previously invoked so that the default repository is available. That is, the first time you invoke SAS/EIS, a default repository is automatically created under the SASUSER library. (A repository is a library managed by the Common Metadata Repository (CMR) that contains information about data tables and columns. A repository does not contain the actual tables; instead, it contains data about the tables.) △

Exporting Metadata from SAS/Warehouse Administrator

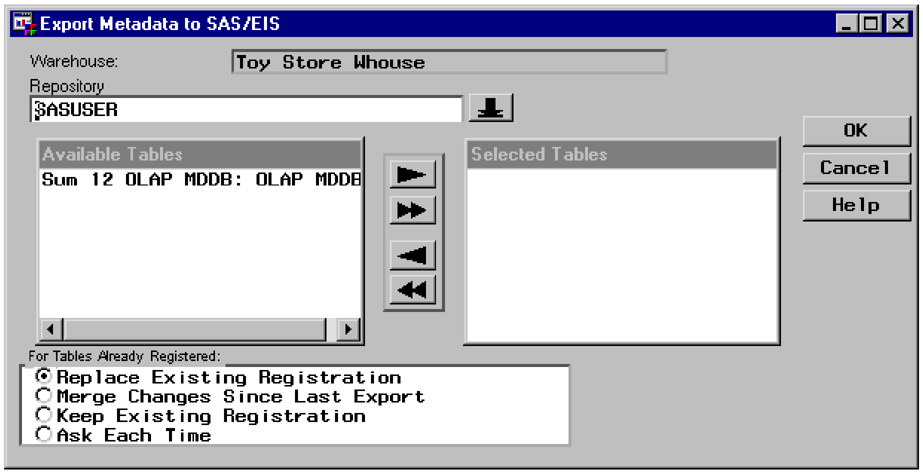
In the SAS/Warehouse Administrator Explorer, position the cursor on the data store for which you want to export metadata. This example exports an OLAP MDDB that is grouped in an OLAP Group of type MOLAP. Even though you cannot export an OLAP Group of type MOLAP or ROLAP, exporting from the group lets you export multiple objects if appropriate. Therefore, for this example, select the OLAP Group **`MOLAP Group`**.



Click the right mouse button, and select **`Export Metadata`**. The Export Format Selection window displays:



In the Export Format Selection window, select **`SAS/EIS Metabase`** as the export format, and then click OK. The Export Metadata to SAS/EIS window displays:


Export Metadata to SAS/EIS

Warehouse: Toy Store Whouse

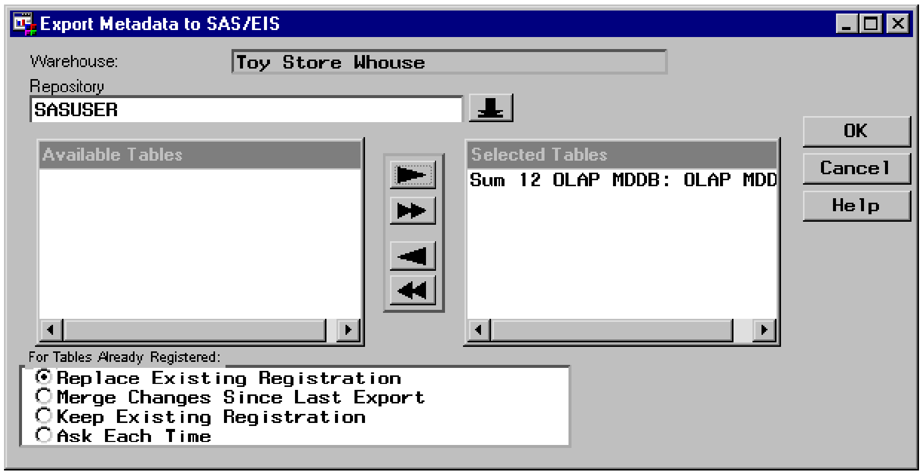
Repository: SASUSER

Available Tables: Sum 12 OLAP MDDB: OLAP MDDB

Selected Tables:

For Tables Already Registered:
- (●) Replace Existing Registration
- ( ) Merge Changes Since Last Export
- ( ) Keep Existing Registration
- ( ) Ask Each Time

Under **Available Tables**, select the OLAP MDDB **Sum 12 OLAP MDDB**, and select the right arrow to move it to **Selected Tables**.


Export Metadata to SAS/EIS

Warehouse: Toy Store Whouse

Repository: SASUSER

Available Tables:

Selected Tables: Sum 12 OLAP MDDB: OLAP MDD

For Tables Already Registered:
- (●) Replace Existing Registration
- ( ) Merge Changes Since Last Export
- ( ) Keep Existing Registration
- ( ) Ask Each Time

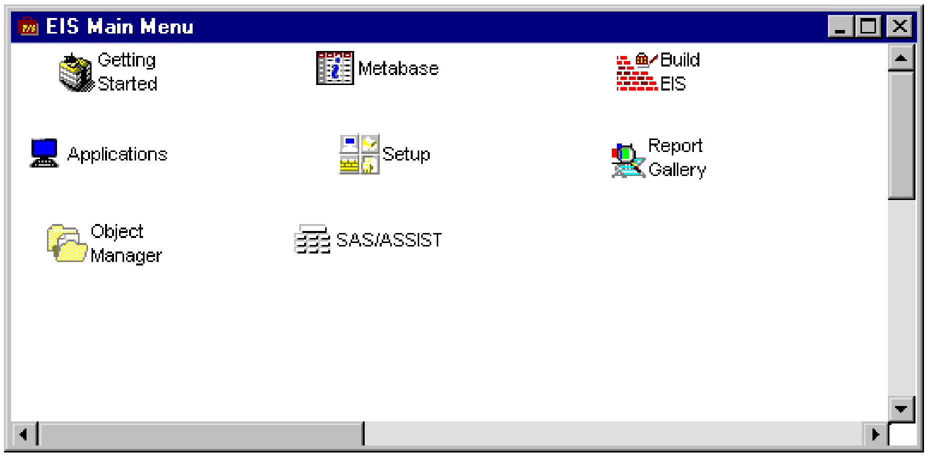
Click OK.

The metadata for the OLAP MDDB is exported from SAS/Warehouse Administrator and automatically registered in the default repository for SAS/EIS. When the export is complete, you will receive a confirmation message to which you click OK.

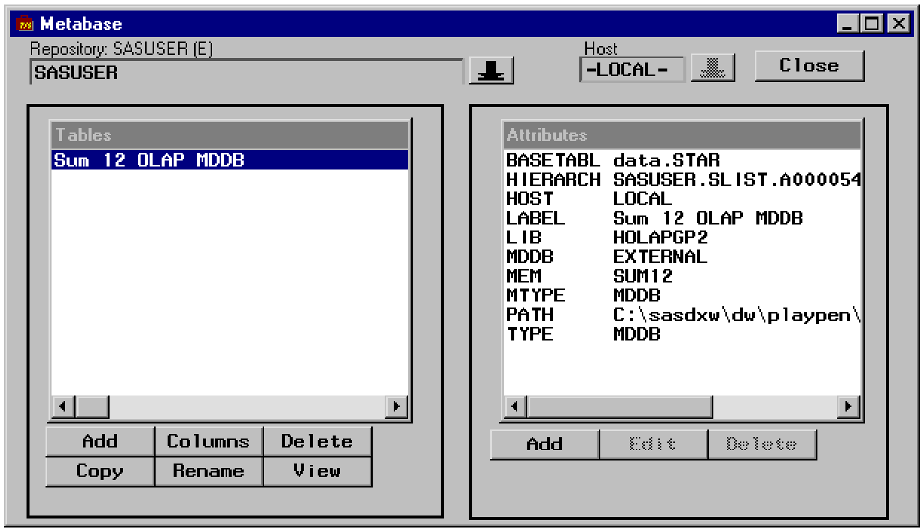
Note: When you export metadata for an OLAP Group of type HOLAP to SAS/EIS, in addition to automatically registering the metadata in a repository, distributed multidimensional metadata is also automatically registered. That is, you do not need to issue the MDMDDB command in SAS/EIS to register the definition. △

Accessing Metadata in SAS/EIS Software

To access the exported and registered metadata, first invoke SAS/EIS software, which displays the EIS Main Menu:



Double-click the **`Metabase`** icon, which opens the Metabase window. The window displays the active repository, which for this example is the default repository SASUSER, and then lists the registered files.



Repository: SASUSER (E)

SASUSER

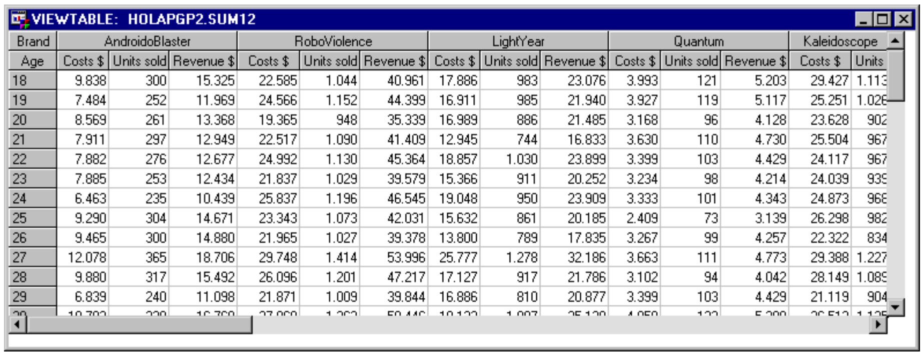
Tables

Sum 12 OLAP MDDB

Attributes

BASETABL	data.STAR
HIERARCH	SASUSER.SLIST.A000054
HOST	LOCAL
LABEL	Sum 12 OLAP MDDB
LIB	HOLAPGP2
MDDB	EXTERNAL
MEM	SUM12
MTYPE	MDDB
PATH	C:\sasdxw\dw\playpen\
TYPE	MDDB

You can view the OLAP MDDB by selecting the table, then clicking View:



VIEWTABLE: HOLAPGP2.SUM12

Brand	AndroidoBlaster			RoboViolence			LightYear			Quantum			Kaleidoscope	
Age	Costs $	Units sold	Revenue $	Costs $	Units sold	Revenue $	Costs $	Units sold	Revenue $	Costs $	Units sold	Revenue $	Costs $	Units
18	9.838	300	15.325	22.585	1.044	40.961	17.886	983	23.076	3.993	121	5.203	29.427	1.113
19	7.484	252	11.969	24.566	1.152	44.399	16.911	985	21.940	3.927	119	5.117	25.251	1.026
20	8.569	261	13.368	19.365	948	35.339	16.989	886	21.485	3.168	96	4.128	23.628	902
21	7.911	297	12.949	22.517	1.090	41.409	12.945	744	16.833	3.630	110	4.730	25.504	967
22	7.882	276	12.677	24.992	1.130	45.364	18.857	1.030	23.899	3.399	103	4.429	24.117	967
23	7.885	253	12.434	21.837	1.029	39.579	15.366	911	20.252	3.234	98	4.214	24.039	939
24	6.463	235	10.439	25.837	1.196	46.545	19.048	950	23.909	3.333	101	4.343	24.873	968
25	9.290	304	14.671	23.343	1.073	42.031	15.632	861	20.185	2.409	73	3.139	26.298	982
26	9.465	300	14.880	21.965	1.027	39.378	13.800	789	17.835	3.267	99	4.257	22.322	834
27	12.078	365	18.706	29.748	1.414	53.996	25.777	1.278	32.186	3.663	111	4.773	29.388	1.227
28	9.880	317	15.492	26.096	1.201	47.217	17.127	917	21.786	3.102	94	4.042	28.149	1.089
29	6.839	240	11.098	21.871	1.009	39.844	16.886	810	20.877	3.399	103	4.429	21.119	904

PART 5

Appendices

APPENDIX

1

Converting Metadata for Environments and Warehouses

Overview

If you have used SAS/Warehouse Administrator Release 1.1, 1.2, or 1.3 to create Data Warehouse Environments, you must convert the metadata repositories in these old Environments to Release 2.0 format in order to use these Environments in SAS/ Warehouse Administrator Release 2.0. A Metadata Conversion wizard is provided that will convert Release 1.x metadata to Release 2.0 metadata.

To invoke the Metadata Conversion wizard, run SAS/Warehouse Administrator Release 2.0, add a new Environment to the desktop. In the `Path` field for the new Environment, specify the path to a Release 1.x Environment's metadata repository that has not been converted to Release 2.0 format. The Metadata Conversion wizard will display.

The Metadata Conversion wizard

- copies the Environment metadata repository (libname _MASTER) for a Release 1.x Warehouse Environment that you specify
- copies the Warehouse metadata repositories (libname _DWMD) for each Data Warehouse within the Environment
- creates Release 2.0 metadata repositories for the new Environment and for each of its Data Warehouses in the locations and with the options that you specify; by default, the new repositories are SAS Version 7 libraries
- creates a default process flow and Job for each data store with its own LOAD step in the original Environment.

The Metadata Conversion wizard operates under the following rules:

- The wizard can only be invoked from the SAS/Warehouse Administrator Release 2.0 interface, as previously described.

- The destination directories that you specify for the converted metadata repositories must exist on the file system. The wizard will not create them for you.
- The destination directories must be different from the ones storing the original metadata.
- The destination directories that you specify for the converted metadata repositories must be empty.

Note: The conversion wizard does not alter the old Environment. △

The wizard copies the old metadata, converts the copy to Release 2.0 format, and saves the Release 2.0 metadata to the locations that you specify.

Note: The conversion wizard only converts the Environment metadata repository (libname _MASTER) and the Warehouse metadata repositories (libname _DWMD) for each Data Warehouse within the Environment. △

The converted Warehouse Environment and its Data Warehouses will point to the data stores, code libraries, and other resources as they are specified in the metadata for the original Environment. For details, see "Verifying Local Resources in the Converted Environment" on page 327.

Note: You cannot use the Release 2.3 SAS/Warehouse Administrator Metadata API to access metadata from previous releases. △

If such access is attempted, when the _SET_PRIMARY_REPOSITORY_ method is called, a message will appear indicating that the metadata must first be converted. The only way to convert the metadata is to invoke the Metadata Conversion wizard, as previously described.

Creating a Directory Structure for the New Environment

The Metadata Conversion wizard requires you to specify directories for the converted Warehouse Environment and for each of its Data Warehouses. The wizard will not automatically create these directories. Accordingly, you might want to create these directories before starting the Metadata Conversion wizard. Keep the following in mind as you create these directories:

- The destination directories must be different from the ones storing the original metadata.
- The destination directories that you specify for the converted metadata repositories must be empty.
- The Warehouse Environment metadata repository should have a unique pathname, and each Data Warehouse metadata repository should have a unique pathname.

Here is an example of a valid directory structure for an Environment (_env) and two Data Warehouses:

```
.\Project-2\_env
.\Project-2\_wh1
.\Project-2\_wh2
```

Later, you might want to add directories for any local data stores, code libraries, or other local resources that will eventually be part of the new Warehouse Environment. These directories are not required for metadata conversion, however.

Inspecting the Pathname for the Old Environment

In the metadata for the Warehouse Environment to be converted, the physical path for the metadata repository might be entered as a relative pathname. That is, the path for the metadata library _MASTER can be entered with a pathname such as **`.\Project-1\_env`** rather than a fully qualified pathname such as **`D:\Dw_projects\Project-1\_env`**. In order for the Metadata Conversion wizard to handle relative pathnames properly, start SAS in a directory that will resolve the pathname to the metadata library _MASTER .

For example, suppose that the pathname for a particular Warehouse Environment is **`.\Project-1\_env`**. To resolve this relative pathname, the current directory shown at the lower right of the SAS window would have to be the parent directory for the **`Project-1`** directory.

Before running the Metadata Conversion wizard, inspect the pathname of the Warehouse Environment that you want to convert. Here is one way to do that.

1 Using the old version of SAS that was used to create the Warehouse Environment to be converted, start SAS/Warehouse Administrator as if you were going to work with that Environment.

   The icon for the Environment to be converted should appear on the desktop.

2 Position the cursor on the Environment, click the right mouse button and select **`Properties`**.

   The Data Warehouse Environment Properties window displays.

3 Inspect the pathname in the Path field.

If the pathname in an Environment **`Path`** field is a relative path, when using the Metadata Conversion wizard, start SAS in a directory that will resolve the pathname to the metadata library _MASTER .

If the pathname in an Environment **`Path`** field is a fully qualified path, when using the Metadata Conversion wizard, you can start SAS in any convenient directory.

Converting a Release 1.x Environment and Its Warehouses

Here is one way to convert the metadata repositories in a Warehouse Environment and its Data Warehouses. Some of the steps provide instructions for dealing with relative pathnames, which you might or might not have.

1 Start SAS/Warehouse Administrator Release 2.0.

2 Verify that the current directory shown at the lower right of the SAS window is appropriate for the Warehouse Environment that you want to convert to Release 2.0 format. If it is not, change the current directory as appropriate. For details, see “Inspecting the Pathname for the Old Environment” on page 325.

   For example, suppose that the correct parent directory for a given Warehouse Environment is **`D:\Dw_projects\`**.

3 On the SAS/Warehouse Administrator desktop, position the cursor in an empty area, click the right mouse button, and select

   Add Item ▶ Data Warehouse Environment

   The Data Warehouse Properties Window displays.

4 In the **`Path`** field, use the right arrow to select the directory associated with the metadata repository for the Warehouse Environment that you want to convert to Release 2.0 format.

For example, suppose that the fully qualified pathname for a particular Warehouse Environment is **`D:\Dw_projects\Project-1\_env`**.

5 Click OK.

The Metadata Conversion wizard displays.

6 Click Next.

The Metadata Conversion Read Me File tab displays.

7 When you are finished reading the Read Me tab, click Next.

The Environment Information tab displays.

8 On the Environment Information tab, verify that the **`Path`** for the Environment and its Data Warehouses will resolve in the current directory shown at the lower right of the SAS window. For example, if the current directory is **`D:\Dw_projects\`**, the relative paths **`.\Project-1\_env`** and **`.\Project-1\_wh1`** will resolve.

9 When finished with the Environment Information tab, click Next.

The Metadata Destination tab for the new Environment displays.

10 On the Environment Metadata Destination tab, click Browse to select the directory for the new metadata repository (libname _MASTER). This directory must already exist, or you will have to create it now. For details, see "Creating a Directory Structure for the New Environment" on page 324.

For example, for an Environment, you might select a path such as **`D:\NewDw_projects\Project-2\_env`**.

11 When finished with this tab, click Next.

The Metadata Destination tab for a Data Warehouse within the new Environment displays.

12 On the Warehouse Metadata Destination tab, click Browse to select the directory for the new metadata repository (libname _DWMD). This directory must already exist, or you will have to create it now.

For example, for a Warehouse, you might select a path such as **`D:\NewDw_projects\Project-2\_wh1`**.

13 When finished with this tab, click Next.

The Metadata Destination tab for any remaining Data Warehouses within the new Environment displays. Repeat the step above for all Data Warehouses in the current Environment.

14 When finished with the Warehouse Metadata Destination tab, click Next.

The Begin Metadata Conversion tab displays.

15 Verify that the information on the Conversion tab is correct, and then click Finish.

The old Environment metadata repository is copied, and new metadata repositories for the Environment and its Data Warehouses are created in the locations that you specified. An icon for the new Environment is added to the SAS/Warehouse Administrator desktop.

See the SAS log for any errors and for a detailed record of what occurs during the metadata conversion. If the SAS log shows no errors, the new Warehouse Environment is ready to use in SAS/Warehouse Administrator Release 2.0.

Unless you specified otherwise, the new metadata repositories are SAS Version 7 libraries. They contain Environment metadata or Warehouse metadata appropriate for SAS/Warehouse Administrator Release 2.0. The converted Warehouse Environment and its Data Warehouses will point to the data stores, code libraries, and other resources as they are specified in the metadata for the original Environment. Default Process Flows and Jobs will have been created for all data stores that had Process Flows in the original Environment.

Opening a Converted Environment for the First Time

To test the converted Environment, open it in the SAS/Warehouse Administrator Explorer. The steps below assume that the SAS/Warehouse Administrator Release 2.0 desktop is displayed and that there is an icon for the new Environment on the desktop.

1 Verify that the current directory shown at the lower right of the SAS window is appropriate for the new Warehouse Environment—the one in Release 2.0 format. If it is not, change the current directory as appropriate. For details, see "Inspecting the Pathname for the Old Environment" on page 325.

   For example, suppose that the correct parent directory for a converted Warehouse Environment is **`D:\NewDw_projects\`**.

2 Position the cursor on the new Environment, click the right mouse button, and select **`Edit`**.

   The Environment should open in the SAS/Warehouse Administrator Explorer. You should be able to display its Data Warehouses and other resources normally.

The Release 2.0 Explorer has not changed much from previous releases. For details about the Explorer, click its window, then from the main menu, select

Help ▶ Using This Window

Verifying Local Resources in the Converted Environment

As previously noted, the converted Warehouse Environment and its Data Warehouses will point to the data stores, code libraries, and other resources *as they are specified in the metadata for the original Environment*. This is especially significant for local resources.

Local Resources with Fully Qualified Pathnames

If the pathname for a local resource is specified as a fully qualified pathname in the original Warehouse Environment, the converted Environment will use the same resource as the original Environment. SAS/Warehouse Administrator Release 2.0 can use resources that were registered in Release 1.x Warehouse Environments. No updates are necessary to make them work in Release 2.0.

For example, suppose that a R1.x Environment resides in the directory **`D:\r13_projects\_env`**. Suppose also that the global metadata for the R1.x Environment specifies a source code library that has a fully qualified pathname of **`D:\r13_projects\_src_code`**. When you convert the R1.3 Environment to a R2.0 Environment, the path for the new Environment might be **`D:\r20_projects\_env`**. However, the global metadata for the R2.0 Environment still points to the source code library at **`D:\r13_projects\_src_code`**.

You might want your new R2.0 Environment to point to resources that reside in locations that were convenient for the old R1.3 Environment. If you do not want to use an old local resource that is specified in the converted metadata, you could

□ copy the old resource to an appropriate local directory for the new Environment. Update the Release 2.0 metadata for that resource so that it points to the new location.

□ create a new local resource and update the Release 2.0 metadata for that resource so that it points to the new location.

Local Resources with Relative Pathnames

If the pathname for a local resource is specified as a relative pathname in the original Warehouse Environment, the converted Environment will attempt to resolve the pathname relative to the current directory shown at the lower right of the SAS window. It is possible that a relative path that worked in the old Environment will not work in the new Environment. To address that issue, you could do either of the following:

- Copy the old resource to an appropriate local directory for the new Environment. Update the Release 2.0 metadata for that resource so that it points to the new location.
- Create a new local resource. If necessary, update the Release 2.0 metadata for that resource so that it points to the new location.

If the Release 2.0 metadata specifies the location of local resources with relative pathnames, and if you create a directory structure for the new Environment that is similar to the directory structure for the old Environment, you might not have to update the metadata for the local resources.

For example, suppose that a Release 1.x Environment included a SAS library definition whose pathname was specified as **`.\_lib-1`**. If the directory structure that you create for the Release 2.0 Environment included a **`.\_lib-1`** directory, you might not have to update the metadata for this library in order for it to be accessible in the Release 2.0 Environment. If you want to access the old content at the new location, copy the contents of the old directory to the new directory. If you want to create new content in the new directory, leave the new directory empty for now. Later, in SAS/Warehouse Administrator Release 2.0, you can specify new information to be written to the **`.\_lib-1`** library.

Process Flows in the New Process Editor

Overview

After an Environment has been converted with the Metadata Conversion wizard, and you have verified that the Environment's metadata is pointing to the correct local data stores, code libraries, or other local resources, the processes for its tables should work as they did before the conversion. However, before you try executing processes in the new Process Editor, keep the following in mind:

- The Release 2.0 Process Editor has changed considerably from previous releases. For details about the Process Editor, click its window, then from the main menu, select

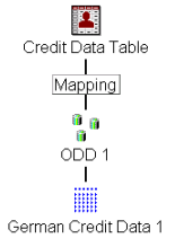
  Help ▶ Using This Window

- The Metadata Conversion wizard creates a default process flow and Job for each data store with its own LOAD step in the original Environment. As a result, complex Process Flow diagrams in your old Environment will be separated into their constituent diagrams—one for each data store with a LOAD step.
- In Release 2.0, the steps for executing a process flow have changed. Details are provided in the next section.

What the Converted Process Flows Look Like

The Metadata Conversion wizard creates a default process flow and Job for each data store with its own LOAD step in the original Environment. For example, suppose that before metadata conversion, the process flow for Credit Data Table looked like the one in Display A1.1 on page 329.

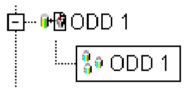
Display A1.1 Release 1.x Process Flow



In a Release 1.x process flow, each icon (with the exception of inputs to ODDs) has its own load step. Accordingly, in the previous display, ODD 1 and Credit Data Table each have their own LOAD steps.

After running the Metadata Conversion wizard, there would be a Job and a data flow for ODD 1, and there would be a separate Job and data flow for Credit Data Table. For example, after metadata conversion, if you opened ODD 1 in the Release 2.0 Process Editor, the following pair of items would be displayed in the left panel:

Display A1.2 ODD 1 Release 2.0 Job Hierarchy

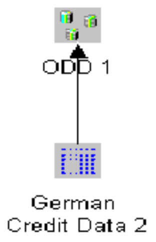


In Display A1.2 on page 329, the item with the rectangle around it is the output table for the Job. The Job is represented by the parent icon of the output table.

A Job is a new Process Editor object in Release 2.0. It is a metadata record that specifies the processes that create one or more data stores. The processes can be specified with a process flow diagram in the Process Editor. If a process flow diagram is specified, SAS/Warehouse Administrator can generate code for the Job. Alternatively, a Job can reference a user-supplied program that contains the processes that create the data store(s). A Job can include scheduling metadata that enables the process flow or user-supplied program to be executed in batch mode at a specified date and time.

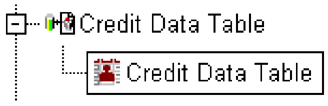
In the Process Editor, to the right of the Job and its output table, any process flow associated with the current Job is displayed, as shown in Display A1.3 on page 330.

Display A1.3 ODD 1 Release 2.0 Process Flow

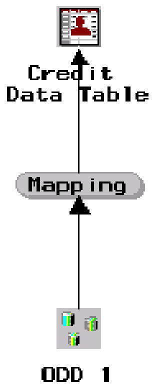


Credit Data Table would have its own Job and process flow, as shown in Display A1.4 on page 330 and Display A1.5 on page 330.

Display A1.4 Credit Data Table Release 2.0 Job Hierarchy



Display A1.5 Credit Data Table Release 2.0 Process Flow



Note that in Display A1.5 on page 330, ODD 1 appears as an input to a mapping process that feeds Credit Data Table. The Credit Data Table Job created by the Metadata Conversion wizard will not generate the code that creates and loads ODD 1. ODD 1 is assumed to be loaded and available for the Credit Data Table Job. In order to create Credit Data Table, then, you would execute the ODD 1 Job shown in Display A1.2 on page 329, then execute the Credit Data Table Job shown in Display A1.4 on page 330.

Note: In Release 2.0, you can specify multiple output tables in one Job as long as the metadata for the output tables is within the same repository (within the same Data Warehouse, for example). △

The Metadata Conversion wizard does not automatically combine all of the data stores (output tables) in a given process flow into one Job because

- each data store can only be created by one Job
- the same data store can be used in several different Jobs
- the site administrator must decide which Job should create the data store, and which Jobs should assume that this data store has already been created.

Also, not all of the metadata for the data stores in a given process flow is in the same repository. For example, in Display A1.5 on page 330, the metadata for ODD 1 is stored at the Environment level (in the _MASTER repository), while the metadata for Credit Data Table is stored at the Data Warehouse level (in the current _DWMD repository).

For details about specifying multiple output tables in one Job, see the "Maintaining Jobs" chapter in this document.

Testing Converted Process Flows

One way to test a Release 1.x process flow after it has been converted to Release 2.0 is to execute each Job interactively.

Note: Load the input tables first, then the output tables. △

For example, in order to test the Credit Data Table process flow, you would execute the ODD 1 Job shown in Display A1.2 on page 329, then execute the Credit Data Table Job shown in Display A1.4 on page 330. Here are the general steps for executing process flows in Release 2.0.

1. If you have not done so already, open the converted Environment in SAS/ Warehouse Administrator Release 2.0.

   For details, see "Opening a Converted Environment for the First Time" on page 327.
2. In the Explorer, expand the relevant groups until the table whose process flow you want to execute is displayed.
3. In the Explorer, position the cursor on the table whose process flow you want to execute, click the right mouse button, and select **`Process`**.

   The table you selected will be opened in the Process Editor. The left panel of the Process Editor will contain a Job and an output table for the table you selected, similar to Display A1.2 on page 329. The right panel will contain the process flow for the table you selected, similar to Display A1.3 on page 330.
4. In the left panel (Job Hierarchy view), position the cursor on the Job for the table whose process flow you want to execute, click the right mouse button, and select **`Run`**.

   The Load Generation/Execution Properties window displays.
5. On the Load Generation/Execution Properties window, click $\boxed{\text{Submit}}$.

   The process flow will be executed.
6. To verify that the process flow executed successfully, review the SAS log and check the output of the process on the file system.

If the process flow fails to execute because a local data store, code library, or other local resource cannot be found, verify that the metadata for this object in the Release 2.0 Environment is pointing to the correct location. For details on this issue, see "Verifying Local Resources in the Converted Environment" on page 327.

APPENDIX 2 Adding the Example Environment

Overview

When you install SAS/Warehouse Administrator software, an example Warehouse Environment called *Marketing Warehouse* is also installed. This appendix describes how to add the Marketing Warehouse Environment to the SAS/Warehouse Administrator desktop.

You can use the Marketing Warehouse Environment to view the kind of metadata that is specified for various kinds of groups, data stores, and processes. This can help you understand what kind of metadata you must enter for similar objects in your data warehousing project.

PC Host Instructions

1 Execute **!SASROOT\whouse\sasmisc\windemo.exe**. The default location the files will be restored to is **c:\temp\dwdemo**.
2 After the files have been restored, the following directory structure will be created:

```
c:\temp\dwdemo
c:\temp\dwdemo\_dwmd
c:\temp\dwdemo\_master
c:\temp\dwdemo\fmts
c:\temp\dwdemo\jobinfo
c:\temp\dwdemo\master_jobinfo
c:\temp\dwdemo\null
c:\temp\dwdemo\sdata
c:\temp\dwdemo\src
c:\temp\dwdemo\wdata
```

3 This demo contains formats that are referenced by placing the following libname statement in your autoexec file:

```
libname library 'dwdemo\fmts';
```

4 Invoke Version 8 of SAS.
5 To add the sample Environment, change the current working directory to the directory where the sample is installed. On the SAS command line, type:

```
x cd c:\temp\
```

Where **c:\temp\** is the main directory where the example Environment is installed. This statement can also be put in your autoexec file.

6 Type **dw** on the command line to invoke SAS/Warehouse Administrator software.

7 After the Desktop window initializes, select the **Add Item** menu option and then the **Data Warehouse Environment** menu option from the **File** pull-down menu.

8 In the Data Warehouse Environment Properties window, type the following in the **Path** field:

```
dwdemo/_master
```

Leave the other fields as they are and click OK.

9 The sample warehouse is now added to the desktop. Double-click the Environment's icon to begin exploring it.

Note: If you receive the following message

```
ERROR: Library _MASTER does not exist
```

then the current working directory is not set correctly. △

For instructions about how to set the current working directory, see Step 2. You will need to exit SAS/Warehouse Administrator software and re-invoke it to have the current working directory modifications take effect.

UNIX Host Instructions

1 Restore the **!SASROOT/misc/warehouse/unxdemo.tar** file to the location where the example Environment will reside. In these instructions, this location will be referred to as **/usr/dw**. If you do not know the location of **!SASROOT**, please contact your SAS representative.

2 Change to the main directory for the sample Environment: **cd /usr/dw**

3 Uncompress the sample Environment: **tar -xvf !SASROOT/misc/warehouse/unxdemo.tar**

4 After the files have been restored, the following directory structure will be created:

```
./dwdemo/
./dwdemo/_dwmd
./dwdemo/_master
./dwdemo/fmts
./dwdemo/jobinfo
./dwdemo/master_jobinfo
./dwdemo/null
./dwdemo/sdata
./dwdemo/src
./dwdemo/wdata
```

5 This demo contains formats that are referenced by placing the following libname statement in your autoexec file:

```
libname library 'dwdemo\fmts';
```

6 Invoke Version 8 of SAS.

7 To add the sample Environment, change the current working directory to the directory where the sample is installed. On the SAS command line, type

```
x cd /usr/dw
```

Where **`/usr/dw`** is the main directory in which the example Environment is installed. This statement can also be put in your autoexec file.

8 Type **`dw`** on the command line to invoke SAS/Warehouse Administrator software.

9 After the Desktop window initializes, select the **`Add Item`** menu option and then the **`Data Warehouse Environment`** menu option from the **`File`** pull-down menu.

10 In the Data Warehouse Environment Properties window, type the following in the **`Path`** field:

```
dwdemo/_master
```

Leave the other fields as they are and click OK.

11 The sample warehouse is now added to the desktop. Double-click the Environment icon to begin exploring it.

Note: If you receive the following message

```
ERROR: Library _MASTER does not exist
```

then the current working directory is not set correctly. △

For instructions about how to set the current working directory, see Step 2. You will need to exit SAS/Warehouse Administrator software and re-invoke it to have the current working directory modifications take effect.

APPENDIX 3

Customizing the SAS/Warehouse Administrator Interface

Overview

Customizing the SAS/Warehouse Administrator interface requires writing new SAS Component Language code (SCL code) or modifying existing SCL code. For information about SCL, refer to *SAS Component Language: Reference*, Version 7 or later.

Add-In Tools

Add-in tools are SCL applications that extend the functionality of SAS/Warehouse Administrator. If any add-ins have been installed in your copy of SAS/Warehouse Administrator, you can display them as follows:

1 Open an Environment in the SAS/Warehouse Administrator Explorer.
2 From the main menu, select

Tools ▶ Add-Ins

A list of add-ins will be displayed.

Add-in tools are used to customize SAS/Warehouse Administrator for your site. You can write your own tools, or you can obtain the tools that have been developed by SAS. To obtain the add-in tools that have been developed by SAS, see "Customizing SAS/Warehouse Administrator" on page 17. Use this section as a reference when writing your own add-in tools.

Customizing the Add-In Tools Registry

The Add-In Tools Registry is a SAS data set that contains references to any add-in tools that have been installed for a given copy of SAS/Warehouse Administrator. When you install add-in tools that have been developed by SAS, the registry is updated by the installation routine. If you develop your own add-in tool, you must update the Add-In Tools Registry so that your tool will be available in SAS/Warehouse Administrator.

The purpose of the tool registry is to provide greater flexibility in the entry that is called, as well as the parameters that the entry is called with. The registry data set is contained in the SASHELP library and has a name of WATOOLS. Any site-specific add-ins can be installed into the SASHELP.WATOOLS, or can be installed as _SASWA.WATOOLS. If the _SASWA.WATOOLS data set is found, its entries and not those of SASHELP.WATOOLS will be used.

Registry Format

The Add-In Tools Registry has the following format:

MNEMONIC
: is an indicator of which frame the SCL program is being called from. Examples:
  - USERTOOL (called from the Explorer)
  - USERPRCS (called from the Process Editor)
  - USERSETP (called from the Setup Frame)

ENTRY
: is the four-level name of the entry to call. This entry can be of any type that can be called using the SCL CALL DISPLAY function.

NAME
: is the name of the tool or tool group. This would be the name shown on the secondary pop-up list when the Tools menu option is selected.

ACTIVE
: is an integer variable, which indicates whether a tool is currently active. A value of zero or missing would indicate that the tool is currently inactive. Any other value indicates that the tool is active. This is provided for entries to remain in the table, but in a deactivated state.

PARMFMT
: is an integer variable indicating the type of parameter list expected by the add-in:
  - 0 — The add-in expects no parameters. There is no entry statement in the called program, or the entry statement contains optional parameters. This parameter format is consistent with Release 1.1
  - 1 — The tool expects 1 parameter. This parameter is a list of named items that allow information to be passed from SAS/Warehouse Administrator to the called add-in.

    L_PARMS — The list of named items that allow information to be passed from the called add-in. The current list of named items that can be passed are
    - REFRESH — A returned numeric value that indicates that SAS/Warehouse Administrator should refresh the displayed screen upon return from the add-in. If a value of 1 is returned, SAS/Warehouse Administrator will refresh the screen as appropriate upon return. Any other value will be ignored and no refresh will be performed.

        - MNEMONIC — A character item that is passed to the add-in indicating the context in which the add-in was called. The value corresponds to the allowable mnemonics as documented under the registry format.

    The entry statement for the called tool should be coded as follows:

```
entry l_parms 8;
```

- 2 — The tool uses the Metadata API and thus expects the following three parameters.
    - ID — The metadata ID of the object that is currently selected in the active SAS/Warehouse Administrator frame. If no object is currently active, this field contains a blank value.
    - I_API — An initialized instance of the metadata API object.

      Note: The called application should not terminate the API object passed. It will be terminated when control is returned to the SAS/Warehouse Administrator. △

    - L_PARMS — The list of named items that allow information to be passed from the called add-in. See the previous paragraphs for a more detailed description of the contents of this list.

  The entry statement for the called tool should be coded as follows:

```
entry id $ 26 I_api 8 l_parms 8;
```

DESC
: is a 200-character field that will contain a description of the entry in this row. It will not be shown through the user interface but is merely for information purposes when editing the table.

Example Add-In Tools Registry

Here is an example that clarifies how tool registry parameters affect the availability of add-in tools available from the Add-Ins pull-down menu. Suppose that the SASHELP.WATOOLS data set contained the information in the following table.

Table A3.1 Example Add-In Tools Registry

MNEMONIC	ENTRY	NAME	ACTIVE	PARM-FMT	DESC
USERTOOL	_SASWA.DW.USERTOOL.SCL	User-Written Tools	1	0	
USERSETP	SASHELP.ADMIN.WATOOLS.FRAME	Administration Tools	1	1	
USERTOOL	SASHELP.DWEXPLT.IMPORT.FRAME	Import Data Model	1	2	

MNEMONIC	ENTRY	NAME	ACTIVE	PARM-FMT	DESC
USERTOOL	SASHELP.DWEXPLT.EXPORT.FRAME	Export Data Model	0	2	
USERPRCS	_SASWA.DW.USERPRCS.SCL	User Written Tools	1	0	

When the

File ▶ Tools ▶ Add-Ins

pull-down menu option is selected from the Explorer, a query will be issued against the data set with an alias USERTOOL and Active=1. If more than one row is found to match the query, a secondary pop-up list is shown that contains the names of the registered add-in tools. For example,

- User-Written Tools
- Import Data Model

Note that the Export Data Model add-in is not included because it is marked as Inactive. If only one row is found, then no pop-up list is displayed. When you select an add-in tool, a search is performed for the corresponding catalog entry. If the entry is not found, an error message is displayed stating that the add-in could not be found. If the entry is found, it is called by using a CALL DISPLAY SCL function.

Importing Column Metadata

The Columns tab on the Table Properties frame is used to define column metadata for ODDs, detail tables, and other tables. SAS/Warehouse Administrator lets you import column metadata from various sources, including SAS data sets and views, the supplied data location, ODDs, detail tables, PROC CONTENTS output, and COBOL file descriptors. You can also add your own list of import tools to the catalog entry SASHELP.DW.USERIMPT.SCL.

Select

Edit ▶ Import ▶ Other

from the pull-down menu or click Import and then select

Other

on the Columns tab to drive the first USERIMPT.SCL entry found in the search path. By default, SASHELP.DW.USERIMPT.SCL displays a message that says that there are no user-defined metadata import tools available. To replace this message with a list of your own tools, put your own entry in USERIMPT.SCL that points to the catalog entries for your tools. The recommended catalog.library for this new entry is _SASWA.DW.

Predefined Column Formats

Your column metadata import tools can be written to pass back a list of columns of a predefined format. SAS/Warehouse Administrator will convert that list into column metadata. This will allow you, for example, to define the metadata for a particular kind of detail table by selecting a user-defined tool from the

Edit ▶ Import ▶ Other

selection on the Edit menu or from the

Other

selection displayed by clicking Import.

To accomplish this, you must define the parameter **l_addcols** in the SCL source code for your metadata import tool. **l_addcols** will contain one or more sublists: one sublist for each column in the table you want to create. Each sublist must have the following format:

```
(NAME=column-name               $8
 DESCRIPTION=column-description  $200
 TYPE=column-type                $1 (C or N)
 LENGTH=column-length            8 (positive integer)
 FORMAT=column-format            $32
 INFORMAT=column-informat        $32
)
```

If your metadata import tool uses the **l_addcols** parameter to define a set of column formats, SAS/Warehouse Administrator will add this information to the column metadata displayed on the Columns tab.

The only named item required in the column sublist is NAME.

NAME must be a valid SASNAME, 1 to 32 characters in length. If the NAME named item is not on the column sublist, an error message will be displayed and that column sublist will be ignored. However, SAS/Warehouse Administrator will continue trying to add the other columns in the **l_addcols**.

If the name supplied is a duplicate of a name already registered in column metadata, the "Duplicate Name" dialog box will be displayed asking for a different name for the column.

Processing will continue after a valid name is provided or that column will not be added if CANCEL is selected on the dialog box.

If DESCRIPTION, FORMAT, and INFORMAT are not provided, they will be defaulted to _BLANK_. If TYPE is not provided, it will default to C. If LENGTH is not provided, it will default to 8 if type is N or 200 if type is C.

If the TYPE specified is not a valid value (C or N), the default C will be used. If the LENGTH specified is not a valid length (positive integer), the type-dependent default will be used.

User-Defined Formats and Informats

On the Columns and Column Roles tabs, when you select the down control widget, the default pop-up lists displayed for formats and informats are as follows:

```
For FORMAT:     Type C:       SASHELP.FSP.FMTC.HELP
                Type N:       SASHELP.FSP.FMTN.HELP
                Unknown Type:   SASHELP.FSP.FMT.HELP

For INFORMAT:   Type C:       SASHELP.FSP.IFMTC.HELP
                Type N:       SASHELP.FSP.IFMTN.HELP
                Unknown Type:   SASHELP.FSP.IFMT.HELP
```

To provide your own pop-up list, SAS/Warehouse Administrator searches for the following entry names in the search path:

```
For FORMAT:     Type C:       USERFMTC
                Type N:       USERFMTN
```

```
                    Unknown Type:   USERFMT

For INFORMAT:       Type C:       USERIFMC
                    Type N:       USERIFMN
                    Unknown Type:   USERIFM
```

The entry type for any of these user-defined entries can be either HELP, LIST, or MENU. (The SCL command that will be used is LISTC.) The value returned should be the selected format or informat character string.

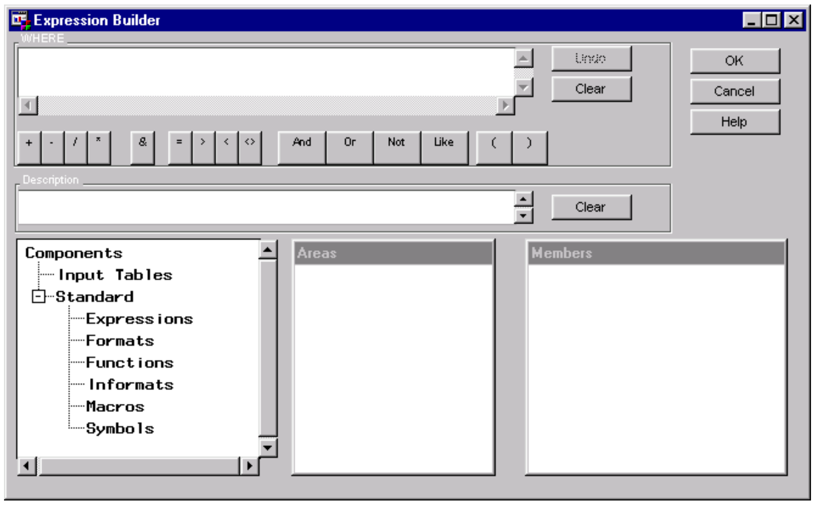
These same pop-up lists are also used in the pop-up lists on the Add Column and Define Statistics for Analysis Columns frames (the frames driven from the Column Roles tab). The Expression Builder frame for defining derived mappings does not use these user-written pop-up lists.

Customizing the Components List for the Expression Builder Window

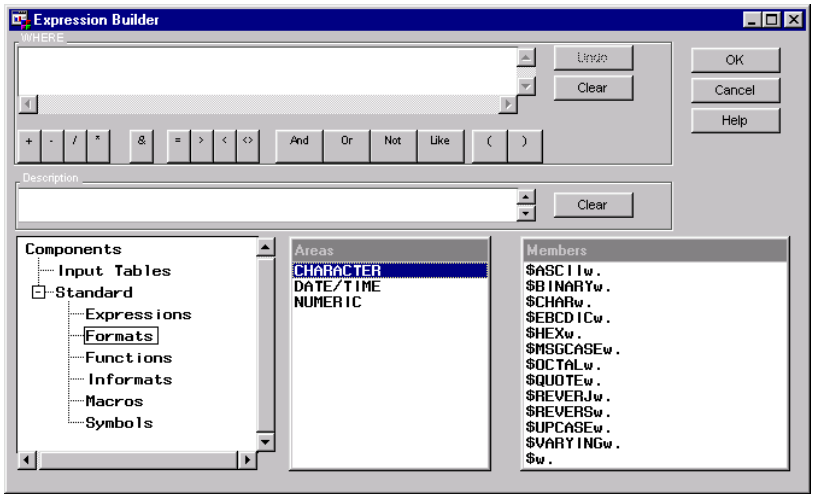
Overview

In SAS/Warehouse Administrator, you use the Expression Builder window to define a SAS expression, which can transform columns, provide conditional processing, calculate new values, and assign new values. For example, you could use the Expression Builder window when defining properties for a Mapping process to convert a character date into a SAS date.

In the Expression Builder window, under the label **`Components`**, is a list of expression components from which you select as you define an expression.



SAS/Warehouse Administrator obtains the default list of expression components that display under the label **`Standard`** from the SAS data set SASHELP.WAXFORM. For example, the following Expression Builder window displays the default character formats:



To add your own expression components, you can create a SAS data set named _SASWA.WAXFORM, using the same format as SASHELP.WAXFORM. When _SASWA.WAXFORM exists, the Expression Builder window displays the label **Extensions**, which lists the customized expression components, for example, **My Informats** and **My Macros**.

Note: For more information about the Expression Builder window and how to define expressions, see the online Help that displays by clicking Help . △

Creating _SASWA.WAXFORM

To add your own expression components, create a SAS data set named _SASWA.WAXFORM, using the same format as SASHELP.WAXFORM. The format of SASHELP.WAXFORM is as follows:

Table A3.2 SASHELP.WAXFORM Format

Column Name	*Type*	*Length*	*Format*	*Informat*	*Label*
category	C	40	$40.	$40.	Primary categorization of expression component.
area	C	40	$40.	$40.	Secondary categorization of the expression component that is specific to the category.
name	C	40	$40.	$40.	Name of expression component.
desc	C	200	$200.	$200.	Description of expression component.
text	C	198	$198.	$198.	Text of expression component.

Column Name	*Type*	*Length*	*Format*	*Informat*	*Label*
preblank	N	8	BEST12.	BEST32.	Indicates whether a blank should be inserted before the text for this transformation. Less than 1 = no preceding blank. 1 or more = add a preceeding blank.
pstblank	N	8	BEST12.	BEST32.	Indicates whether a blank should be inserted after the text for this transformation. Less than 1 = no succeeding blank. 1 or more = add a succeeding blank.
usedesc	N	8	BEST12.	BEST32.	A variable used to determine if the description field entered for this component should replace the description for the expression. 1 replaces the description. 0 says that the description is for informational purposes.
active	N	8	BEST12.	BEST32.	Indicates whether this entry is active. 1 = active entry and should appear as a component. 0=not active.

When _SASWA.WAXFORM exists, the Expression Builder window displays the label **`Extensions`**, which lists the customized expression components. The items in the components list are taken from the list of unique values of the CATEGORY column in _SASWA.WAXFORM.

Note the following when you create _SASWA.WAXFORM:

- The TEXT column value can contain double quotes (") or single quotes ('), but not both.
- The combination of data set, category, area, and name makes a row unique. Therefore, the same area name can be contained in multiple categories, and the same name can be contained in multiple areas.
- You can insert placeholders into the TEXT column to signify the location and type of parameters needed. The following is a list of placeholders:

Table A3.3 Text Column Placeholders

Placeholder	*Meaning*
%character%	character value
%numeric%	numeric value
%datetime%	SAS datetime value
%date%	SAS date value
%year%	2 or 4 character year

Placeholder	*Meaning*
%quarter%	the number 1-4 to represent the quarters of a year
%month%	the number 1-12 to represent the month of the year
%day%	the number 1-31 to represent the day of the month
%interval%	a character string to represent the type of interval to use
%hour%	the number 0-23 to represent the hour of the day
%minute%	the number 0-59 to represent the minute of an hour
%second%	the number 0-59 to represent the second of the minute
%juliandate%	julian date value

APPENDIX

4

Metadata Export Reference

Overview

This appendix describes the format of metadata that is exported with the Export Format Selection window. It also describes how to write your own add-in metadata exporters.

SAS Data Sets Exported for Shared Metadata

"Exporting Host Definitions and Other Shared Metadata" on page 316 describes how to export the metadata defined for SAS libraries, hosts, DBMS connections, and contacts. This section describes the format of the exported data sets.

```
GENERAL:
    #    Variable    Type    Len    Pos    Label
  ---------------------------------------------------------------------
    1    INFOTYPE    Char      8      0    Information Type
    2    OBJNAME     Char     40      8    Name
    3    OBJTYPE     Char     40     48    Type
    4    OBJ_ID      Char     17     88    ID
    5    VERSION     Char      8    105    Version of Metadata
    6    DESC        Char    200    113    Description
    7    PARENTNM    Char     40    313    Owning Parent Name
    8    PARENTTP    Char     40    353    Owning Parent Type
    9    PARENTID    Char     17    393    Owning Parent Metadata ID
   10    PARENTLB    Char      8    410    Owning Parent Metadata Libref
   11    MDLIBREF    Char      8    418    Metadata Libref
   12    MDCREATE    Char     20    426    Metadata Created
   13    MDMODIFY    Char     20    446    Metadata Modified
   14    DTCREATE    Char     20    466    Data Created
   15    DTMODIFY    Char     20    486    Data Modified
   16    NOTES       Char      3    534    Notes Available
```

```
        17    NOTESLOC    Char     40    537    Notes Location

   LIBRARY:
         #    Variable    Type    Len    Pos    Label
             -----------------------------------------------------------
         1    INFOTYPE    Char      8      0    Information Type
         2    OBJNAME     Char     40      8    Name
         3    OBJTYPE     Char     40     48    Type
         4    OBJ_ID      Char      8     88    ID
         5    VERSION     Char      8     96    Version of Metadata
         6    LOCNAME     Char     40    104    Library Name
         7    LOCDESC     Char    200    144    Library Description
         8    ASSGNBY     Char     30    344    Assigned by
         9    LIBREF      Char      8    374    Libref
        10    ENGINE      Char      8    382    Engine
        11    PATH        Char     40    390    Physical Path
        12    LIBOPTS     Char    200    430    Options

   HOST:
         #    Variable    Type    Len    Pos    Label
             -----------------------------------------------------------
         1    INFOTYPE    Char      8      0    Information Type
         2    OBJNAME     Char     40      8    Name
         3    OBJTYPE     Char     40     48    Type
         4    OBJ_ID      Char      8     88    ID
         5    VERSION     Char      8     96    Version of Metadata
         6    HOST        Char     40    104    Host Name
         7    LOCALE      Char     10    144    Locale
         8    REMOTE      Char    200    154    Remote Address
         9    SCRIPT      Char    200    354    Remote Script
        10    COMAMID     Char      8    554    Access Method

   DBMSCONN:
         #    Variable    Type    Len    Pos    Label
             -----------------------------------------------------------
         1    INFOTYPE    Char      8      0    Information Type
         2    OBJNAME     Char     40      8    Name
         3    OBJTYPE     Char     40     48    Type
         4    OBJ_ID      Char      8     88    ID
         5    NICKNAME    Char      8     96    DBMS Nickname
         6    SCHEMA      Char     32    104    User/Schema
         7    PROTECT     Char      3    136    Password Protected
         8    LASTMOD     Char     20    139    Password Last Modified
         9    DBMSOPTS    Char    200    159    Connection Options

   CONTACT:
          #    Variable    Type    Len     Pos    Label
             ------------------------------------------------------------
          1    INFOTYPE    Char      8       0    Information Type
          2    OBJNAME     Char     40       8    Name
          3    OBJTYPE     Char     40      48    Type
          4    OBJ_ID      Char      8      88    ID
          5    VERSION     Char      8      96    Version of Metadata
          6    PERSNAME    Char    200     104    Person Name
```

```
     7    TITLE1      Char    200     304    Title (Primary)
     8    TITLE2      Char    200     504    Title (Secondary)
     9    PHONE1      Char     20     704    Phone (Primary)
    10    PHONE2      Char     20     724    Phone (Secondary)
    11    EMAIL       Char     40     744    Email Address
    12    ADDR1       Char    200     784    Address (line 1)
    13    ADDR2       Char    200     984    Address (line 2)
    14    ADDR3       Char    200    1184    Address (line 3)
    15    ADDR4       Char    200    1384    Address (line 4)

EXTATTRS:
     #    Variable    Type    Len     Pos    Label
      ------------------------------------------------------------
     7    DESC        Char    200     313    Extension Description
     1    INFOTYPE    Char      8       0    Information Type
     2    OBJNAME     Char     40       8    Name
     3    OBJTYPE     Char     40      48    Type
     4    OBJ_ID      Char     17      88    Metadata ID
     6    VALUE       Char     200     113   Extension Value
     5    VERSION     Char      8      105   Version of Metadata
```

SAS Data Sets Exported for Groups and Data Stores

"Exporting Metadata for Groups and Data Stores" on page 316 describes how to export metadata for any object displayed in the Explorer window except an Environment object. This section describes the format of the exported data sets.

```
GENERAL:
    #    Variable    Type    Len    Pos    Label
  ---------------------------------------------------------------------
    1    INFOTYPE    Char      8      0    Information Type
    2    OBJNAME     Char     40      8    Name
    3    OBJTYPE     Char     40     48    Type
    4    OBJ_ID      Char     17     88    ID
    5    VERSION     Char      8    105    Version of Metadata
    6    DESC        Char    200    113    Description
    7    PARENTNM    Char     40    313    Owning Parent Name
    8    PARENTTP    Char     40    353    Owning Parent Type
    9    PARENTID    Char     17    393    Owning Parent Metadata ID
   10    PARENTLB    Char      8    410    Owning Parent Metadata Libref
   11    MDLIBREF    Char      8    418    Metadata Libref
   12    MDCREATE    Char     20    426    Metadata Created
   13    MDMODIFY    Char     20    446    Metadata Modified
   14    DTCREATE    Char     20    466    Data Created
   15    DTMODIFY    Char     20    486    Data Modified
   16    SUMTYPE     Char     20    506    Summary Table Type
   17    NUMCHILD    Char      8    526    Number of Direct Subordinates
   18    NOTES       Char      3    534    Notes Available
   19    NOTESLOC    Char     40    537    Notes Location

SUPPORT:
    #    Variable    Type    Len     Pos    Label
        ------------------------------------------------------------
    1    INFOTYPE    Char      8       0    Information Type
    2    OBJNAME     Char     40       8    Name
    3    OBJTYPE     Char     40      48    Type
    4    OBJ_ID      Char     17      88    Metadata ID
    5    VERSION     Char      8     105    Version of Metadata
    6    ROLE        Char     20     113    Support Role
    7    PERSNAME    Char    200     133    Person Name
    8    TITLE1      Char    200     333    Title (Primary)
    9    TITLE2      Char    200     533    Title (Secondary)
   10    PHONE1      Char     20     733    Phone (Primary)
   11    PHONE2      Char     20     753    Phone (Secondary)
   12    EMAIL       Char     40     773    Email Address
   13    ADDR1       Char    200     813    Address (line 1)
   14    ADDR2       Char    200    1013    Address (line 2)
   15    ADDR3       Char    200    1213    Address (line 3)
   16    ADDR4       Char    200    1413    Address (line 4)

LOCATION:

    #    Variable    Type    Len     Pos    Label
    --------------------------------------------------------------------
    1    INFOTYPE    Char      8       0    Information Type
    2    OBJNAME     Char     40       8    Name
    3    OBJTYPE     Char     40      48    Type
    4    OBJ_ID      Char     17      88    Metadata ID
    5    VERSION     Char      8     105    Version of Metadata
    6    LOCROLE     Char     20     113    Location Role
    7    LOCNAME     Char     40     133    SAS Library Name
    8    LOCDESC     Char    200     173    SAS Library Description
    9    ASSGNBY     Char     30     373    Assigned by
   10    LIBREF      Char      8     403    SAS Libref
   11    ENGINE      Char      8     411    SAS Library Engine
   12    PATH        Char     40     419    SAS Library Path
   13    OPTIONS     Char    200     459    SAS Library Options
   14    STORFORM    Char      8     659    Storage Format
   15    TABLE       Char      8     667    Table Name
   16    HOST        Char     40     675    Host Name
   17    LOCALE      Char     10     715    Locale
   18    REMOTE      Char    200     725    Remote Address
   19    SCRIPT      Char    200     925    Remote Script
   20    COMAMID     Char      8    1125    Access Method
   21    PROTECT     Char      3    1133    Protected
   22    TBLTYPE     Char     40    1136    DBMS Table Type
   23    CONNNAME    Char     40    1176    DBMS Connection Name
   24    CONNICK     Char      8    1216    DBMS Connection Nickname
   25    CONNUSER    Char     32    1224    DBMS Connection User/Schema
   26    CONNOPTS    Char    200    1256    DBMS Connection Options

COLUMNS:

    #    Variable    Type    Len    Pos    Label
```

```
         -----------------------------------------------------------
    1    INFOTYPE    Char      8      0    Information Type
    2    OBJNAME     Char     40      8    Name
    3    OBJTYPE     Char     40     48    Type
    4    OBJ_ID      Char     17     88    Metadata ID
    5    VERSION     Char      8    105    Version of Metadata
    6    COLNAME     Char      8    113    Column Name
    7    COLDESC     Char    200    121    Column Description
    8    COLTYPE     Char      1    321    Column Type
    9    LENGTH      Char      8    322    Column Length
   10    FORMAT      Char     32    330    Column Format
   11    INFORMAT    Char     32    362    Column Informat
   12    SUMROLE     Char     20    394    Summary Role
   13    CNOTELOC    Char     35    414    Column Note

FISCAL:

    #    Variable    Type    Len    Pos    Label
         -----------------------------------------------------------
    1    INFOTYPE    Char      8      0    Information Type
    2    OBJNAME     Char     40      8    Name
    3    OBJTYPE     Char     40     48    Type
    4    OBJ_ID      Char     17     88    Metadata ID
    5    STOD        Char      9    105    Start Time of Day
    6    SDOW        Char      9    114    Start Day of Week
    7    SDOM        Char      9    123    Start Day of Month
    8    SMOY        Char      9    132    Start Month of Year
```

Add-In Metadata Exporters

The Export Metadata Facility allows user-supplied add-in exporters to be called from inside SAS/Warehouse Administrator. A SAS Metadata API application can be registered and subsequently called from the SAS/Warehouse Administrator Export Format Selection window. The application can access the metadata through the API and export that metadata using the desired method.

The Add-In Exporter feature is designed much like the Add-In Tools Registry. The Add-In Exporter Registry is contained in a SAS data set. The default data set contains the entries that register the existing SAS supplied exporter tools. This data set resides in the SASHELP library and has a name of WAEXPRT. Any site-specific add-ins can be installed into SASHELP.WAEXPRT, or can be installed as _SASWA.WAEXPRT. If the _SASWA.WAEXPRT data set is found, its entries are used and not those of the SASHELP.WAEXPRT data set.

Usage

The Add-In Exporter Registry has the following format:

MNEMONIC	is an indicator of which frame the export tool is being called from: USERTOOL — called from the Explorer USERSETP — called from the Setup Frame
ENTRY	is the four-level name of the entry to call. This entry can be of any type that can be called using the SCL CALL DISPLAY function.

NAME	is the name of the exporter tool. This is the name shown as a selection option in the Export Format Selection window.
ACTIVE	is an integer variable, which indicates whether this exporter is currently active. A value of zero or missing indicates that the exporter is currently inactive. Any other value indicates the tool is active. This is provided so entries can remain in the table even though deactivated.
DESC	is a 200-character field that contains a description of the entry in this row. It will not be shown in the user interface but is merely for information purposes when editing the table.

The Add-In Exporter uses the SAS Metadata API and will pass the following three parameters to the entry specified in the registry when that exporter is selected:

ID
: is the Metadata ID of the object that is currently selected in the active SAS/Warehouse Administrator frame. If no object is currently active, this field contains a blank value.

I_API
: is an initialized instance of the Metadata API object.

  Note: The called application should not terminate the API object passed to it. The API object will be terminated when control is returned to the SAS/Warehouse Administrator. △

L_PARMS
: is an empty SCL list that can be used to pass return code information from the called application back to SAS/Warehouse Administrator. The L_PARMS list is designed to contain named items that pass information back and forth between SAS/Warehouse Administrator and the called application. The only named item in the list that SAS/Warehouse Administrator currently processes is RC, which is expected to be a list.

  If you choose to pass return code information back to SAS/Warehouse Administrator (which will produce a pop-up error message), your application should create a list with two named items, 'RC' and 'MSG'. The 'RC' item is a numeric return code, the 'MSG' item is a character string that will become the message in the pop-up error message. Add this list you create to L_PARMS as the named item 'RC' to return the value to SAS/Warehouse Administrator. A non-zero RC value will signal an error to SAS/ Warehouse Administrator.

  The entry statement for the called tool should be coded as follows:

```
entry id $ 26 i_api 8 l_parms 8;
```

Example Add-In Application

Suppose you have written a Metadata API application that uses the API to access information about a Detail Table, and the application writes the pertinent metadata about that Detail Table and its process to an external file. You want to make this tool available from the Explorer in SAS/Warehouse Administrator, and you want to disable the SAS data set exporter that is provided for the Explorer. To do this, you add the necessary information to the SASHELP.WAEXPRT data set, as follows:

Table A4.1 Example Add-In Exporter Registry

Mnemonic	Entry	Name	Active	Desc
USERTOOL	SASHELP.DW.EXPAEIS.SCL	SAS/EIS Metabase	1	
USERTOOL	SASHELP.DW.EXPASAS.SCL	SAS Datasets	0	
USERSETP	SASHELP.DW.EXPASASG.SCL	SAS Datasets	1	
USERTOOL	YOURLIB.YOURAPI.APP.FRAME	My Exporter	1	

After this modification is made to the SASHELP.WAEXPRT data set, your changes will be reflected in the Export Format Selection window. When you select an item in the Explorer and choose to Export Metadata, you will have the option of selecting SAS/EIS Metabase or My Exporter. The SAS Datasets option will no longer appear because you have set active=0.

If you were to choose My Explorer, SAS/Warehouse Administrator will invoke YOURLIB.YOURAPI.APP.FRAME with a CALL DISPLAY SCL statement. If the entry is not found, an error message will be displayed.

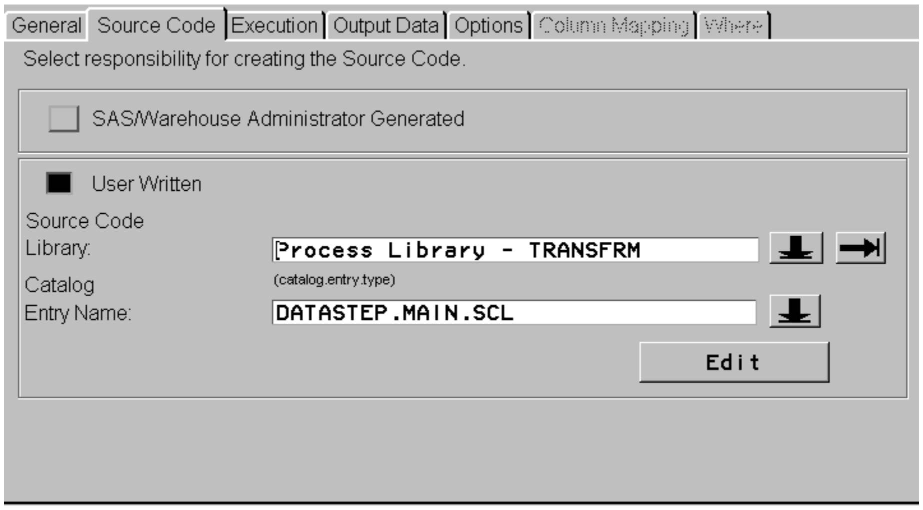
APPENDIX 5 Add-in Code Generators and the Process Library

Add-In Code Generator Technical Reference

An *add-in code generator* is a SAS Metadata API application that is registered in SAS/Warehouse Administrator to dynamically generate the code for a process step. These applications enable you to access and use relevant metadata to drive your table processes.

After an add-in code generator is written and stored in the proper SAS library, you can specify it as a user-written routine on the Source Code tab for a given process step. For example, in Display A5.1 on page 356, an add-in code generator named DATASTEP.MAIN.SCL has been specified for a process.

Display A5.1 Source Code Tab With Add-in Code Generator Specified



Select responsibility for creating the Source Code.

SAS/Warehouse Administrator Generated

User Written

Source Code Library: Process Library - TRANSFRM

(catalog.entry.type)

Catalog Entry Name: DATASTEP.MAIN.SCL

The difference between specifying a source entry for your source code and specifying an SCL entry is that the source entry gets included in the Job that is created, while the SCL entry you specify gets run during the creation of that Job. The SCL entry you supply should create the source code that will actually be run when the Job is submitted.

The add-in code generator that you specify is passed parameters that make it context-sensitive and metadata-aware. The code you provide can use the Metadata API to dynamically create the code for that process step.

For details about the Metadata API, see *SAS/Warehouse Administrator Metadata API Reference, Release 2.3*.

Requirements

There are certain requirements you must adhere to in order to successfully take advantage of an add-in code generator. When using this feature you are no longer only providing code to be run to create your table, you are actually integrating your API application with SAS/Warehouse Administrator, and that integration must follow the rules described in this section.

SAS/Warehouse Administrator will pass these parameters, in the following order, to your application:

ID
: is the metadata ID of the process object associated with the Process Editor object you have selected.

I_API
: is an initialized instance of the Metadata API object. This instance will already be initialized with the primary and possibly secondary repositories, as appropriate for the object selected.

  Note: The called application should not terminate the API object passed. It will be terminated when control is returned to SAS/Warehouse Administrator. △

L_PARMS
: is an empty SCL list that can be used to pass return code information from the called application back to SAS/Warehouse Administrator.

  The L_PARMS list is designed to contain named items that pass information back and forth between SAS/Warehouse Administrator and the called application. The only named item in the list that SAS/Warehouse Administrator currently processes is RC, which is expected to be a list. If you choose to pass return code information back to SAS/Warehouse Administrator (which will produce a pop-up error message), your application should create a list with two named items, 'RC' and 'MSG'.

The 'RC' item is a numeric return code, the 'MSG' item is a character string that will become the message in the pop-up error message. Add this list you create to L_PARMS as the named item 'RC' to return the value to SAS/Warehouse Administrator. A non-zero RC value will signal an error to SAS/Warehouse Administrator.

The entry statement for the called add-in code generator should be coded as follows:

```
entry id $ 26 i_api 8 l_parms 8;
```

SAS/Warehouse Administrator uses the preview buffer to store generated code for all the process steps in the Process Editor, including access code for input sources and macro definitions between steps. Your SCL code will need to place any code it generates into the preview buffer in order for that code to be saved or submitted for that step.

CAUTION:

Because the preview buffer is shared by the SAS/Warehouse Administrator generation process and your add-in code generation process, it is critical that you take great care in handling the preview buffer. △

You should write to the preview buffer as necessary, but you should not clear it out (thereby deleting any prior generated code). You should not submit the code in the Process Editor (so do not use submit **`CONTINUE;`** statements).

SAS/Warehouse Administrator will control whether the generated code should be submitted, saved, or viewed, based on how the code generation process was initiated.

Usage Notes

There are several small differences between API applications run as add-in code generators and API applications run as stand-alone. The add-in generator application will always return blank values in the STEP SOURCE CODE and SOURCE CODE properties of any process queried with the _GET_METADATA_ method. The add-in generator code has been registered to generate code that is supplied by those properties, so it cannot request that information even though it creates that information.

An add-in generator API application has been registered in SAS/Warehouse Administrator to generate code for a process and is assumed to be a secure application. The fact that it is a secure application means that API queries of relevant types will return password information about those types.

The WHSASSTR, WHDBMS, and WHLIBRY types have password information about SAS data sets and DBMS connections respectively. If password information is registered for either type, an API query of that object from an add-in code generator application will return the registered user ID and password information.

If the API application is not a secure application, the passwords will be returned as 'XXXXXXXX'.

For details about the values returned for the WHSASSTR, WHDBMS, and WHLIBRY types, see *SAS/Warehouse Administrator Metadata API Reference, Release 2.3*.

CAUTION:

Using the add-in code generator feature demands that you be very aware of the effects your API application will have on SAS/Warehouse Administrator. △

A simple failure to compile your application will cause a Program Halt in SAS/ Warehouse Administrator. Incorrect entry parameters will cause a Program Halt. Any halts or other errors in your application can adversely affect SAS/Warehouse Administrator. Also, take great care with your handling of the preview buffer (see caution note in preceding section). If your application does take a Program Halt for any reason, you should exit SAS/Warehouse Administrator and re-enter it.

Sample Add-In Code Generator

```
*****************************************************************/
/*                                                              */
/*                                                              */
/*              S A S   S A M P L E   L I B R A R Y             */
/*                                                              */
/*                                                              */
/*   TITLE: Data Step Mapping                                   */
/* VERSION: 2.0.0                                               */
/* PRODUCT: SAS/Warehouse Administrator                         */
/*  SYSTEM: Windows 95, Windows NT, Unix                        */
/*    DESC: This program demonstrates how to develop a          */
/*          user-written one-to-one data mapping process by     */
/*          extracting information from existing metadata.  It  */
/*          uses SAS DATA step statements for the column mapping */
/*          to load a detail or data table rather than the      */
/*          PROC SQL code that SAS/WA would generate.           */
*/
/*                                                              */
/*          This add-in retrieves column metadata of the output */
/*          table to do a straight mapping, which selects input */
/*          columns based on the output table metadata.  The    */
/*          column names of the output table remain the same as */
/*          the ones mapped in the input table.                 */
/*                                                              */
/*          This program expects to be passed the ID of a WHPRCMAP*/
/*          type, the process for a mapping in your warehouse.  */
/****************************************************************/


length mpin_lib $ 8;
length mpin_data $ 32;
length mpout_lib $ 8;
length mpout_data mpout_col $ 32
       mpout_coltyp $ 1
       mpout_collen 8
       mpout_colfmt mpout_colinfmt $ 20
       ;
length keepvar $ 32;
length table_id $ 26
       type_id super_type_id $ 8
       ;
```

```
entry id $ 26 i_api 8 l_parms 8;

rc = rc;

INIT:

   /* l_meta is used later to store metadata of */
   /* output tables from data mapping process   */

   l_meta = makelist();
   rc = insertc(l_meta, id, -1, 'ID');

   /* l_var is used later to store column names of the  */
   /* interim data set output from data mapping process */

   l_var = makelist();

   /* get metadata of output table of data mapping process */

   l_outtb = makelist();
   l_meta = insertl(l_meta, l_outtb, -1, 'OUTPUT TABLES');

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

   if l_rc ne 0 then do;
      link MAKERC;
      return;
   end;

   if listlen(l_outtb) < 1 then do;

       rc = 1000;
       rc_msg =
      'ERROR: output table metadata missing';

       link MAKERC;
       return;

   end;

   /* assume only one output table is generated */

   l_outtbA = getiteml(l_outtb, 1);

   /* verify table ID referring to a subtype of WHTBLPRC */

   table_id = getnitemc(l_outtbA, 'ID');
   type_id = scan(table_id, 2, '.');
   super_type_id = 'WHTBLPRC';

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

```
if l_rc ne 0 then do;

   link MAKERC;
   return;

end;

if not a_subtype then do;

   rc = 10001;
   rc_msg = 'ERROR: invalid process ID: ' || id ||
        '.  Expecting an intermediate output table process.';
   link MAKERC;
   return;

end;   /*  if  */

/* extract metadata of the name of the output */
/* table from data mapping process            */

l_outtbA = insertc(l_outtbA, ' ', -1, 'TABLE NAME');

/* extract metadata of the library that the */
/* output table resides                     */

l_outlib = makelist();
l_outtbA = insertl(l_outtbA, l_outlib, -1, 'LIBRARY');

/* extract metadata of input source that serves   */
/* as the input table to data mapping process */

l_insrc = makelist();
l_outtbA = insertl(l_outtbA, l_insrc, -1, 'INPUT SOURCES');

/* extract metadata of output object */

l_outobj = makelist();
l_outtbA = insertl(l_outtbA, l_outobj, -1, 'OUTPUT OBJECTS');

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

if l_rc ne 0 then do;
   link MAKERC;
   return;
end;

*************************************
* information of the input data set *
* used by data mapping process      *
*************************************
;
if listlen(l_insrc) < 1 then do;

   rc = 2000;
   rc_msg =
  'ERROR: input source metadata missing';

   link MAKERC;
   return;

end;

/* one input source is used */

l_insrcA = getiteml(l_insrc, 1);

l_insrcA = insertc(l_insrcA, ' ', -1, 'TABLE NAME');

/* get metadata of the library storing the */
/* input data set for data mapping process */

l_inlib = makelist();
l_insrcA = insertl(l_insrcA, l_inlib, -1, 'LIBRARY');

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

if l_rc ne 0 then do;
   link MAKERC;
   return;
end;

mpin_data = getnitemc(l_insrcA, 'TABLE NAME');

if mpin_data = _blank_ then do;

   rc = 9995;
   rc_msg =
  'ERROR: mapping input data set name missing';
   link MAKERC;
   return;

end;   /*  if  */

else do;

   l_inlib = insertc(l_inlib, ' ', -1, 'LIBREF');

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

   if l_rc = 0 then do;

      mpin_lib = getnitemc(l_inlib, 'LIBREF');

      if mpin_lib = _blank_ then do;
         rc = 9996;
         rc_msg =
     'ERROR: mapping input libref name missing';
         link MAKERC;
```

```
          return;

       end;   /*  if  */

   end;   /*  else  */

   ***************************************
   * information of the interim data set *
   * output from data mapping process    *
   ***************************************
   ;
   /* get the name of the output data set */

   mpout_data = getnitemc(l_outtbA, 'TABLE NAME');

   if mpout_data = _blank_ then do;

      rc = 8887;
      rc_msg =
  'ERROR: mapping output data set name missing';
      link MAKERC;
      return;

   end;   /*  if  */

   /* get metadata of the libref storing the  */
   /* output data set in data mapping process */

   l_outlib = insertc(l_outlib, ' ', -1, 'LIBREF');

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

   if l_rc ne 0 then do;
      link MAKERC;
      return;
   end;

      mpout_lib = getnitemc(l_outlib, 'LIBREF');

      if mpout_lib = _blank_ then do;

         rc = 8888;
         rc_msg =
        'ERROR: mapping output libref name missing';
         link MAKERC;
         return;

      end;   /*  if  */

      submit;
         DATA &mpout_lib.&mpout_data;
      endsubmit;
```

```
end;   /*  if  */


/* get metadata of the output data set columns */

if listlen(l_outobj) < 1 then do;

   rc = 7773;
   rc_msg =
  'ERROR: output object metadata missing';
   link MAKERC;
   return;

end;   /*  if  */

l_outobjA = getiteml(l_outobj, 1);

l_outcol = makelist();
l_outobjA = insertl(l_outobjA, l_outcol, -1, 'COLUMNS');

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

if l_rc ne 0 then do;
   link MAKERC;
   return;
end;

do i = 1 to listlen(l_outcol);

   l_outcolA = getiteml(l_outcol, i);

   l_outcolA = insertc(l_outcolA, ' ', -1, 'NAME');
   l_outcolA = insertc(l_outcolA, ' ', -1, 'TYPE');
   l_outcolA = insertn(l_outcolA, ., -1, 'LENGTH');
   l_outcolA = insertc(l_outcolA, ' ', -1, 'FORMAT');
   l_outcolA = insertc(l_outcolA, ' ', -1, 'INFORMAT');

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

   if l_rc ne 0 then do;
      link MAKERC;
      return;
   end;

   *--------------------*
   * column attributes *
   *--------------------*
   ;
   /* get column name */

   mpout_col =
      getnitemc(l_outcolA, 'NAME');

   if mpout_col = _blank_ then do;
      rc = 7774;
      rc_msg =
    'ERROR: mapping output column name missing';
      link MAKERC;
      return;

   end;   /*  if  */

   /* create an SCL list containing column */
   /* names later referred by the SAS      */
   /* statements in the submit block       */

   l_var = insertc(l_var, mpout_col, -1);

   /* get column type */

   mpout_coltyp =
      getnitemc(l_outcolA, 'TYPE');

   if mpout_coltyp = _blank_ then do;

      rc = 7775;
      rc_msg =
     'ERROR: mapping output column type missing';
       link MAKERC;
       return;

    end;   /*  if  */

   /* get column length */

   mpout_collen =
      getnitemn(l_outcolA, 'LENGTH');

   if mpout_collen <= 0 then do;

      rc = 7776;
      rc_msg =
    'ERROR: mapping output column length missing';
      link MAKERC;
      return;

   end;   /*  if  */

   if mpout_coltyp = 'N' then do;

      /* numeric variable */

      submit;
         LENGTH &mpout_col &mpout_collen;
      endsubmit;

   end;   /*  if  */
```

```
      else if mpout_coltyp = 'C' then do;

         /* character variable */

         submit;
            LENGTH &mpout_col $&mpout_collen;
         endsubmit;

      end;   /*  else  */

      /* get column format */

      mpout_colfmt =
         getnitemc(l_outcolA,'FORMAT');

      if mpout_colfmt ne _blank_ then do;

      /* If column format is specified by   */
      /* the data warehouse administrator,  */
      /* mpout_colfmt will have a non-blank */
      /* value.                             */

         submit;
            FORMAT &mpout_col &mpout_colfmt;
         endsubmit;

      end;   /*  if  */

      /* get column informat */

      mpout_colinfmt =
         getnitemc(l_outcolA,'INFORMAT');

      if mpout_colinfmt ne _blank_ then do;

      /* If column informat is specified by */
      /* the data warehouse administrator,  */
      /* mpout_colinfmt will have a non-    */
      /* blank value.                       */

         submit;
            INFORMAT &mpout_col &mpout_colinfmt;
         endsubmit;

      end;  /*  if  */


   end;   /*  do  */


   submit;
      SET &mpin_lib.&mpin_data
      ( KEEP =
   endsubmit;

   do k = 1 to listlen(l_var);

      keepvar = getitemc(l_var, k);

      submit;
         &keepvar
      endsubmit;

   end;   /*  do  */

   submit;
      );
      RUN;
   endsubmit;

return;

MAKERC:

   passed_rc = rc;

   if listlen(l_rc) = -1 then do;

      l_rc = makelist();
      rc = setnitemn(l_rc, passed_rc, 'RC');

      if rc_msg ne _blank_ then
         rc = setnitemc(l_rc, rc_msg, 'MSG');
   end;

   rc = insertl(l_parms, l_rc, -1, 'RC');

return;

TERM:

   rc = dellist(l_meta, 'Y');
   rc = dellist(l_var, 'Y');

return;
```

Process Library Technical Reference

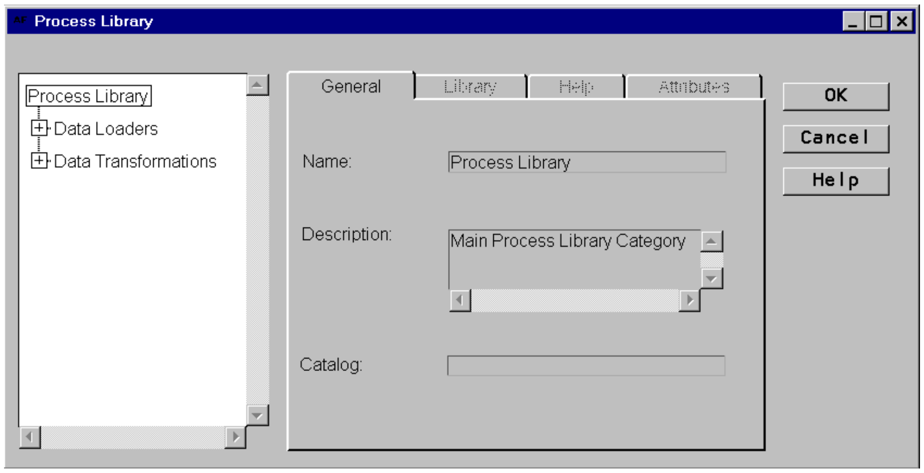
The Process Library is a collection of registered routines that extract, transform, and load data into warehouse tables. As you use the Process Editor to define a step in the data flow for a particular table, you have the option of selecting a pre-defined routine from the Process Library, rather than defining your own process for that step.

The Process Library is made up of a registered set of Process Catalogs. A Process Catalog is a SAS catalog that has a specific set of entries and performs a specific process step. The MAIN entry in a Process Catalog is a reference to the routine that actually performs the step. This routine can be a SOURCE entry or an SCL entry (add-in code generator). For details about the difference between these two kinds of routines, see "Add-In Code Generator Technical Reference" on page 355. For a description of the Process Catalog format, see "Process Catalog Format" on page 372.

Invoking the Process Library

The Process Library window can be invoked using the **`Process Library`** pull-down **`Tools`** menu selection in the Process Editor. If a process step is currently selected, the purpose of calling the Process Library is to define the process information for that particular step. If no step is selected, the Process Library window is called to browse the Process Library. Display A5.2 on page 367 shows a Process Library window.

Display A5.2 Example Process Library Window, Unexpanded



Name: Process Library

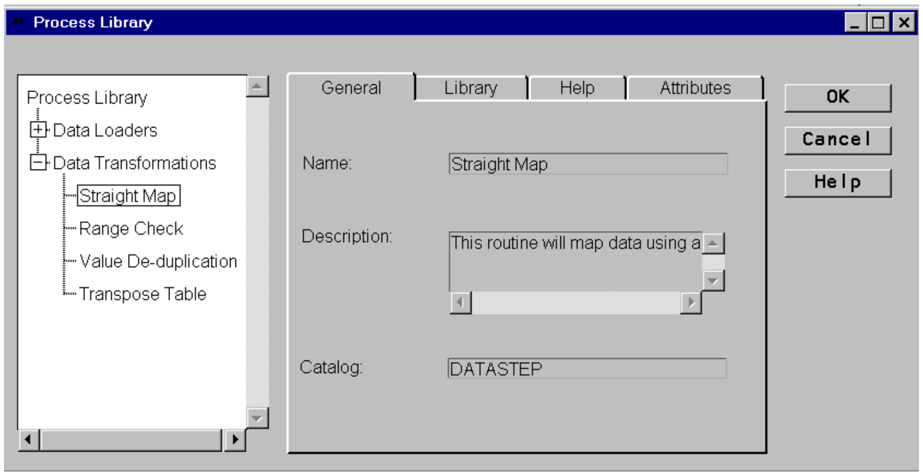
Description: Main Process Library Category

Catalog:

Navigating the Process Library

The Process Library contains a configurable hierarchy of Process Groups and Process Catalogs. A Process Group is a set of related Process Catalogs. In Display A5.3 on page 368, a Process Group (Data Transformations) is expanded, and a Process Catalog (Straight Map Process) is selected. Note the items in the figure are only examples.

Display A5.3 Process Library Window With A Process Selected



Name: Straight Map

Description: This routine will map data using a

Catalog: DATASTEP

The tabbed pages on the right of the display show the metadata about the selected Process Group or Process Catalog. If a Process Group is selected, only the General tab is enabled. If a Process Catalog is selected, all tabs are enabled.

General Tab

The General tab displays descriptive information about the selected object. The tab contains fields for the name (up to 40 characters), a description (up to 200 characters), and, if the selected entry is a process catalog, the catalog name (up to 8 characters).

Note: A Process Catalog naming convention has been adopted to differentiate between Process Catalogs that you define and those that have been supplied by SAS. △

The first letter of the catalog name is restricted to these alphabetic ranges:

User-defined catalogs:
start with a letter between A and R.

Catalogs supplied by SAS Institute:
start with a letter between S and Z.

Library Tab

The Library tab displays library information that is retrieved for the registry for the currently selected Process Catalog. It contains field entries for **`Libref`**, **`Engine`**, **`Path`**, and **`Options`**.

Help Tab

The Help tab displays any note information that is contained in the Process Catalog. You can print this information by positioning the cursor in the white space of the tab, clicking the right mouse button, and selecting **`Print`**.

Attributes Tab

The Attributes tab displays any extended attribute information that is contained in the Process Catalog. In this context, extended attributes are additional items of metadata that the Process Catalog uses to generate the appropriate code for the selected process step. The author of the Process Catalog decides whether to include extended attributes. To print this information, position the cursor in the white space of the tab, click the right mouse button, and select **`Print`**.

Selecting a Process Catalog

When the desired Process Catalog is located, it can be selected by clicking the name of the Process Catalog in the left viewer. If the user desires to select this Process Catalog as the process information for the selected step in the Process Editor, the user can then click $\boxed{\text{OK}}$.

When $\boxed{\text{OK}}$ is selected, the selected Process Catalog information will be interrogated for correctness.

- The library information will be validated by assigning the libref using the supplied library information. If an error occurs while assigning the libref, an appropriate error message will be displayed to the user.
- The metadata for this Data Warehouse Environment will be searched for a Library definition that matches the library information supplied. To determine a match, the libref, engine, path and options information must match exactly, including case, those of an existing library definition.

  If no matching definition is found, the user will be prompted to determine whether a new metadata definition should be created.

  If the user decides to create a new definition, the user will be prompted to enter a name for this new library definition. The default name is the phrase "Process Library -" followed by the supplied libref.
- If the Process Catalog contains an ATTRS.SLIST entry, the attributes will be verified. For any attributes for which an error is found, the user will be prompted to ignore this attribute and continue or to stop the selection.
- If the user is replacing process information that is currently marked as being generated by SAS/Warehouse Administrator, the user will be prompted to confirm the replacement of this process information.

If any errors are discovered, the user selects $\boxed{\text{No}}$ or $\boxed{\text{Cancel}}$ to any confirmation dialogs, and the termination of the frame is halted. To end the frame, the user must then fix the errors, change selections to the confirmation dialogs, or select $\boxed{\text{Cancel}}$.

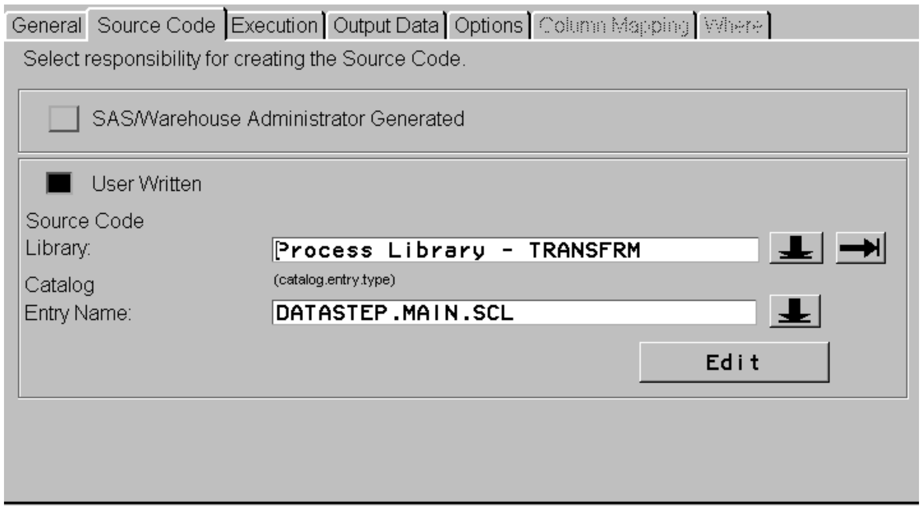
If a Process Group is selected at the time $\boxed{\text{OK}}$ is selected, the process information for the step will not be modified. If $\boxed{\text{Cancel}}$ is selected, the process information for the step will not be modified.

If no step in the Process Editor is selected at the invocation of the Process Library frame, no process information will be modified upon exit of the frame, whether or not $\boxed{\text{OK}}$ or $\boxed{\text{Cancel}}$ is selected.

Selection Results

After selecting a Process Catalog and returning to the Process Editor, the process information for the active step, if any, will have been modified to include a reference to the Process Catalog Main entry. To see this, select **`Properties`** on a process step, or **`Edit Load Step`** on a table. Display A5.4 on page 370 shows the Source Code tab for a process that has been updated by a Process Catalog.

Display A5.4 Source Code Tab After Update by a Process Catalog



Select responsibility for creating the Source Code.

SAS/Warehouse Administrator Generated

User Written

Source Code Library: Process Library - TRANSFRM

(catalog.entry.type)

Catalog Entry Name: DATASTEP.MAIN.SCL

Edit

Any attributes that were registered for the Process Catalog have been added to the metadata as Extended Attributes. These can be seen by selecting **`Extensions`** from the **`File`** pull-down menu option.

Process Library Registry

The display of groups and catalogs are controlled using the WAPRCS data set. If the WAPRCS data set exists in the _SASWA library, this data set will be used. If it is not found in this library, or this library is not assigned, the WAPRCS data set will be used from the SASHELP library. The WAPRCS data set contains one row for every Process group or Process Catalog. The data set contains the following columns:

Table A5.1 WAPRCS Data Set Columns

Col Name	Col Type	Col Length	Col Label	Description
Parent	C	40	Parent Group	Name of the group that contains this group or catalog.
Name	C	40	Name of Catalog or Group	The name of the Process Catalog or Group.
Desc	C	200	Description of Catalog or Group	A brief description of the catalog or group.
Catalog	C	8	Name of SAS catalog	The name of the SAS catalog. This field should be left blank for a Process Group.

Col Name	Col Type	Col Length	Col Label	Description
Active	N	8	1=Row active 0=Row inactive	Whether this row in the data set is active. Only those rows that are active will be used to populate the display.
Libref	C	8	Libref where this catalog resides	The SAS Libref that contains the catalog for this Process Catalog. This field should be left blank for a Process Group.
Engine	C	8	Libname engine of library	The SAS Libname engine used to access this library.
Path	C	200	Libname Path of Library	The host-specific path of the catalog. This corresponds to the path designation on the SAS Libname statement.
Options	C	200	Libname options of Library	The SAS Libname statement options.

Notes

- All names that exist in the PARENT column must match a NAME column in another row. If the PARENT column has a blank value, this entry is the initial level of the hierarchy.
- A row in the data set can depict either a Process Group or a Process Catalog.

  If the NAME of this row exists in the data set as a PARENT, then it is a Process Group.

  If the row is just for a Group, the CATALOG and LIBREF column values should be left blank.

  If the row is a Catalog, at least the CATALOG and LIBREF column values must have values.
- When entering Library information, the following rules apply:

  For a Process Catalog, the LIBREF column must contain a non-blank value.

  If any of the other Library information fields are non-blank (for example, PATH, ENGINE, OPTIONS), this information is used to automatically assign the library when this catalog is selected. If errors are discovered when the library is assigned, the user will be given an appropriate message. This type of definition corresponds to the **`Let the SAS System assign the library`** option in SAS/Warehouse Administrator. When a new metadata definition is created for this library, this option will be selected.
- If only the LIBREF column has a non-blank value (for example, PATH, ENGINE, OPTIONS are blank), this represents a library that is to be assigned by the user. This libref must be assigned by the user at the time the Process Library User

Interface is invoked. This type of definition corresponds to the **`The user will pre-assign the library`** option in SAS/Warehouse Administrator. When a new metadata definition is created for this library, this option will be selected.

- When entering the same library information for several catalogs in the registry data set, it is important to match all of the library information (LIBREF, ENGINE, PATH, OPTIONS) exactly. This includes spacing and case. If the information is not entered in this way, the metadata might be polluted with several library definitions for a single set of library information.

Example Contents for WAPRCS Data Set

The following table lists the contents of an example WAPRCS data set. A Process Library with this data set would look similar to the library shown in the following table:

Table A5.2 Example WAPRCS Contents

Parent	Name	Desc	Active	Catalog	Libref	Path	Engine Options
	Process Library	Main Process Library Group	1				
Process Library	Data Loaders		1				
Process Library	Data Transformations		1				
Data Transformations	Straight Map		1	Datastep	Transfrm	C:\MyPrograms\loaders	
Data Transformations	Range Check		1	Range	Transfrm	C:\MyPrograms\loaders	
Data Transformations	Value De-duplication		1	Dedup	Dataddup	C:\MyPrograms\transforms	BASE Access=Readonly
Data Transformations	Transpose Table		1	Transpos	DataTran		Server=server.shr6

Process Catalog Format

A Process Catalog is a SAS catalog with a defined format. A Process Catalog can contain only a single process. A Process Catalog naming convention has been adopted to differentiate between Process Catalogs that you define and those that have been supplied by SAS.

User-defined catalogs:
start with a letter between A and R.

Catalogs supplied by SAS Institute:
start with a between letter S and Z.

A Process Catalog contains these standard entries:

Main.xxxx (Required) is the main entry point of the catalog. This can either be a SOURCE or SCL type. If it is a SOURCE entry, it will be assumed that the source code should be included as is. If it is an SCL type, the SCL entry will be called during the code generation process. For details, see “Add-In Code Generator Technical Reference” on page 355. The entry name seen on the Source Code tab will be "catalog.MAIN.xxxx".

HELP.SOURCE	(Optional) is a source entry that contains Help information. This Help file will contain information explaining the functionality, any usage notes, and other pertinent information.
ATTRS.SLIST	(Optional) contains the information about the parameters for the process catalog. This entry is an SLIST with a sublist for each parameter (attribute). Each sublist has the following named items: ◻ Name - The name of the parameter (attribute). This name can be up to 40 characters. This item is required for each attribute. ◻ Desc - The description of the parameter (attribute). This can be up to 200 characters in length. This item is optional. ◻ Value - Any initial, default value for the parameter. This named item is optional and if not passed, no default value will be set. See “ATTRS.SLIST Example” on page 373.
BLDATTRS.SCL	(Optional) is recommended that the SCL code that creates the ATTRS.SLIST entry be included in the Process Catalog as BLDATTRS.SCL. See “BLDATTRS.SCL Example” on page 373.

ATTRS.SLIST Example

```
SLIST(
( NAME='FALLBACK' {T}
DESC='Teradata FALLBACK Option' {T}
VALUE='NO' {P}
)[5] {L}
( NAME='ERRLIMIT' {T}
DESC='Teradata ERRLIMIT Option' {T}
)[7] {L}
( NAME='SESSIONS' {T}
)[9] {L}
)[3]
```

In this example, there are three extended attributes for this Process Catalog.

- ◻ For the first attribute, FALLBACK, a description has been given, as well as a default value, NO.
- ◻ For the second attribute, ERRLIMT, a description has been given, but no default value has been specified.
- ◻ For the third attribute, SESSIONS, no description or default value is specified.

Note: It is recommended that a description (item name DESC) and default value (item name VALUE) be included in all attributes.

This will give the user of the Process Library more information to determine what the attribute is used for and, if a default value is supplied, to lessen the amount of manual effort associated with using a process. △

BLDATTRS.SCL Example

```
/**************************************************************
*
```

```
* Create the ATTRS.SLIST entry for this Process Catalog
*
***************************************************************/
init:
/*
* Determine the name of the Process Catalog and build the
* ATTRS.SLIST entry into this catalog.
*/
myname=screenname();
mylib=scan(myname,1,'.');
mycat=scan(myname,2,'.');
attrname=mylib||'.'||mycat||'.ATTRS.SLIST';
/*
* Create the Master List
*/
l_attrs=makelist();
/*
* Create a sublist for each attribute
*/
/*
* Feedback attribute
*/
l_attr=makelist();
l_attrs=insertl(l_attrs,l_attr,-1);
l_attr=insertc(l_attr,'FALLBACK',-1,'NAME');
l_attr=insertc(l_attr,'Teradata FALLBACK Option',-1,'DESC');
l_attr=insertc(l_attr,'NO',-1,'VALUE');
/*
* ERRLIMIT attribute
*/
l_attr=makelist();
l_attrs=insertl(l_attrs,l_attr,-1);
l_attr=insertc(l_attr,'ERRLIMIT',-1,'NAME');
l_attr=insertc(l_attr,'Teradata ERRLIMIT Option',-1,'DESC');
/*
* SESSIONS attribute
*/
l_attr=makelist();
l_attrs=insertl(l_attrs,l_attr,-1);
l_attr=insertc(l_attr,'SESSIONS',-1,'NAME');
/*
* Save the ATTRS.SLIST entry
*/
list_rc=savelist('CATALOG',attrname,l_attrs,0,'Attribute List');
if list_rc ne 0 then do;
msg='ERROR: Saving of '||attrname||' failed with rc='||
trim(left(putn(list_rc,'8.')));
put msg;
end; /* if */
return;
```

APPENDIX

6

Recommended Reading

Recommended Reading

Recommended Reading

Here is the recommended reading list for this title:

- *Cody's Data Cleaning Techniques Using SAS Software*
- *Communications Access Methods for SAS/CONNECT and SAS/SHARE*
- *Moving and Accessing SAS Files*
- *SAS/Warehouse Administrator Metadata API Reference*

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

add-in code generator
is a SAS/Warehouse Administrator Metadata API application that dynamically generates the code for a process in SAS/Warehouse Administrator. It enables you to access and use relevant metadata to drive your table processes. For example, the Teradata Loader add-in reads the metadata for an input table, then generates the code which loads the table into a Teradata DBMS, using Teradata load utilities. If add-in code generators have been installed at your site, they are typically selected from the Process Library.

add-in metadata exporter
is a SAS/Warehouse Administrator Metadata API application that exports SAS/Warehouse Administrator metadata in a format that can be used by another application. For example, the SAS/EIS Metabase Exporter exports metadata in SAS/EIS format. If custom metadata exporters have been installed at your site, they can be selected from the Export Format Selection window.

add-in tool
is a SAS SCL application that performs a task in SAS/Warehouse Administrator. For example, the Publish HTML Documentation tool reads the metadata for an input table and publishes the metadata in HTML format (column descriptions, input sources, source code, and other metdatata). Add-in tools are available from the `Tools-> Add-Ins` pull-down menu.

analysis column
is a numeric column that stores raw data to be used to calculate statistics. The values for an analysis column are the measurements you want to analyze, for example, using OLAP techniques. For example, `Age` and `Income` could be analysis columns.

category column
see *class column*.

cardinality
is the number of discrete values for any OLAP class column within a population.

class column
is a required OLAP summary role column, which is a numeric or character column that is used to group data into classifications. The values for each class column define groups for analysis. That is, the rows in the summary data store are grouped according to the values of the column, and a separate analysis is run for each group.

Class columns can have continuous values, but they typically have a few discrete values that define the classifications of the column. For example, if columns STATE and COUNTY are class columns, you can sort the data so that the states are first, and SAS/Warehouse Administrator will summarize data for each county within each state.

Contact record
is a metadata record which specifies the owner or administrator who is responsible for a group, data store, process, or Job. An *owner* is the person who formulates policy and makes decisions about an object. An *administrator* is the person who implements decisions formulated by the owner in accordance with established policy. Contact records are stored along with the other metadata for the current Environment. Once created, contact records may be included in the metadata records for groups, data stores, processes and Jobs in the current Environment.

crossing
is a unique ordered list of one 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. A crossing represents the physically stored data, which provides the quick response when displaying a report in an OLAP application. For each crossing, there is a single record for each unique combination of all named class columns in the original raw input data.

Data File
a metadata record which specifies a SAS file that is an input to an Operational Data Definition (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 may still want to specify a Process Flow for this Job for documentation purposes.

Data Group
is a simple grouping element for Data Tables, Information Marts, and other Data Groups.

data integration
is a kind of data transformation in which diverse data attributes are reconciled and data values are put into a standard, consistent form. Integration can be performed as a separate task, but is often performed simultaneously with other transformation tasks or during the extraction phase. See also denormalization.

data mart
is a limited data warehouse that is often designed to meet the needs of a particular department or individual. A data mart is more limited in scope than a data warehouse, which typically contains information used by more than one department. To implement a data mart in SAS/Warehouse Administrator, use an appropriate SAS/ Warehouse Administrator object, such as a Data Warehouse or a Data Group.

data mining
is the process of searching for patterns, trends, or correlations in large data stores. Data mining uses statistical analysis techniques, pattern recognition technology, or both. SAS Enterprise Miner software can be used for this purpose.

data preparation process
is a general term for routines which extract, validate, or transform data for loading into a table or file. SAS/Warehouse Administrator data preparation processes include Mappings, Record Selectors, Data Transfers, and User Exits.

data source
is any information that you will extract, transform, and summarize in your warehouse data stores. Data sources can be in any format that SAS can read, on any supported hardware platform.

data store
is a table, a view, or a file that is registered in a Warehouse Environment. Except for ODDs, whether a data store is in SAS format or in another format, you can have SAS/Warehouse Administrator generate the SAS code that is required to read, write, or update the data store. If a file is not a SAS file, you can specify a command that SAS/Warehouse Administrator will use to open the file. Data stores can be in any format that SAS can read and write, on any supported hardware platform.

data structuring
is a kind of data transformation in which new columns are created and/or existing columns are modified to provide users with needed subject information.

data summarization
is a kind of data transformation in which data is aggregated by computing secondary statistics on numeric data or by creating counts on noncontinuous columns.

Data Table
is a metadata record which 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. They can be used to define detail data stores, summary data stores (if you write your own summary code), a look-up table included as part of a join, or a table that holds information that does not fit anywhere else.

Data Transfer process
is a metadata record used to generate or retrieve a routine which moves data from one host to another. Data Transfers are required when an input source and the target data reside on different hosts. SAS/Warehouse Administrator generates code that uses PROC UPLOAD or PROC DOWNLOAD to move the data.

data transformation
is a general term for all of the tasks that prepare data for loading into a data warehouse. See also data integration, data scrubbing, data structuring, data validation, denormalization, and summarization.

data validation
is a kind of data transformation in which potentially invalid, out-of-range, missing, or duplicate values are detected.

data warehouse
in general, a data warehouse is a collection of data that is extracted from one or more sources for the purpose of query and analysis. Data warehouses are subject-oriented, integrated, time-variant, and nonvolatile. In contrast to a data mart, a data warehouse is better suited for larger, more comprehensive information storage.

Data Warehouse
in SAS/Warehouse Administrator, a Data Warehouse object is a metadata record which specifies the SAS library _DWMD. The _DWMD library contains the metadata for most groups and data stores in a data warehouse or a data mart at your site.

Data Warehouse Environment
is a metadata record which specifies the SAS library _MASTER. The _MASTER library contains host definitions and other metadata that is shared among one or more Data Warehouses and ODD Groups. On the SAS/Warehouse Administrator desktop, Environments are displayed as icons. In the SAS/Warehouse Administrator Explorer, the Environment you selected from the desktop is the top-most object.

DBMS
data base management system; in the context of SAS/Warehouse Administrator, this term usually refers to a DBMS other than SAS, such as Oracle, Teradata, or Microsoft Access.

DBMS connection definition
is a metadata record which specifies a user name, a password, DBMS options and other information that SAS requires to access tables in a data base management system (DBMS). Once created, DBMS connection definitions can be included in the metadata records for DBMS data stores in the current Environment. SAS/Warehouse Administrator will then be able to generate the appropriate SAS/ACCESS statements to access these data stores.

DBMS LIBNAME definition
see *SAS/ACCESS LIBNAME definition*.

denormalization
is a kind of data transformation in which DBMS data is freed from its normalized structure by combining information from separate tables. Denormalization can increase the performance of a data warehouse, though it frequently results in some data redundancy.

detail data
is information at or near the fact level in a data base. It is data that has not been summarized or has only been lightly summarized after extraction from a source. In SAS/Warehouse Administrator, detail data is stored in Data Tables, Detail Logical Tables, and Detail Tables.

Detail Logical Table
is a metadata record which specifies a SAS table or view that can serve multiple purposes. Detail Logical Tables are frequently used to define views on multiple, related Detail Tables. They can be also be used to define a detail data store or a simple group for Detail Tables. A Subject can have only one Detail Logical Table. A Detail Logical Table can contain any number of Detail Tables or views. Detail Logical Tables in different Subjects can share (link to) the same Detail Table.

Detail Table
is a metadata record which specifies a SAS table or view or a DBMS table or view that can serve as a detail data store in SAS/Warehouse Administrator. Detail Tables are children of Detail Logical Tables.

dimension
acts as an index for identifying values. A dimension groups related data, which is defined in hierarchies. For example, you could organize sales data into three dimensions: Geography, Time, and Product. The Time dimension could include the following hierarchies, which provide different paths in order to drill down to increasing levels of detail: Time-by-Week and Time-by-Month.

drill-down analysis
is the process of displaying increasing levels of detail within a specified dimension, such as time (from Year to Month to Day) or geographic location (from Country to State/Province to City).

Event
is a metadata record which specifies a condition that controls a Job, such as checking for certain return codes or verifying the existence of a file. Currently, to use Events, you must write a Metadata API program which reads Job Flows with Events and generates code for them.

Environment
See Data Warehouse Environment.

External File
is an input to an ODD which 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.

fact
is an instance of some particular occurrence and the properties of the occurrence all stored in a data base.

global metadata
is metadata that, for the most part, can be shared among one or more Data Warehouses and ODD Groups within an Environment. After you define a Warehouse Environment, you can define metadata records that are shared at the Environment level. These records can be included in the metadata for data stores and other resources in the current Environment. The global metadata types are: SAS library definitions (including SAS/ACCESS LIBNAME definitions and Jobs Information libraries), Host definitions, DBMS connection definitions, Contact records, and Scheduling server definitions.

group
is an element in the SAS/Warehouse Administrator Explorer or Process Editor that is used to organize other elements. Groups in the Explorer include Environments, ODD Groups, Warehouses, and Subjects.

hierarchy
is a unique 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. For example, for a dimension named Time, you could define a hierarchy named Time-by-Month that consists of the class columns YEAR, MONTH, and DATE. Note that the term hierarchy is not the same as a stored summary level, which is a crossing.

HOLAP
is hybrid online analytical processing (OLAP). HOLAP combines the functionality of both ROLAP and MOLAP, producing a hybrid OLAP solution that combines the best features of both. HOLAP provides access to diverse data sources on local and/or remote servers. SAS/Warehouse Administrator supports HOLAP with an OLAP Group of type *HOLAP*. Such a group can contain both OLAP Tables and OLAP MDDBs.

When an OLAP Group of type HOLAP is specified as an output data store in a Job, SAS/Warehouse Administrator generates a *proxy MDDB*, which is a physical file that represents all of the data in an OLAP Group of type HOLAP. The proxy MDDB can be used by SAS/EIS software to provide more efficient access to multiple OLAP Tables and/or OLAP MDDBs.

Host definition
is a metadata record which specifies a computer where data stores reside, where processes and Jobs execute, or where process output is sent. Once created, host definitions may be included in the metadata records for data stores, processes and Scheduling Server definitions in the current Environment.

ID column
is an optional OLAP summary role column that can be used to include additional values in the summary data store. You can have multiple ID columns. Specifying an ID column applies to an OLAP Table only; it is not supported for an OLAP MDDB.

Information Mart (Info Mart)
is a grouping element for Information Mart Items and Information Mart Files.

Information Mart File (Info Mart File)
is a metadata record which specifies a file other than a SAS file that you want to register in a Warehouse. The file can be a spreadsheet, an HTML report, or any file that can be opened by an external application.

Information Mart Item (Info Mart Item)
is a metadata record which contains or displays information generated from detail data or summary data in the Warehouse. These items are usually SAS charts, reports, graphs, or queries.

Job
is a metadata record that specifies the processes that create one or more data stores (output tables). A Job enables you to connect a series of process steps into a single unit. The processes may be specified with a Process Flow diagram in the Process Editor. If a process flow diagram is specified, SAS/Warehouse Administrator can generate code for the job. Alternatively, a job may simply reference a user-supplied program which contains the processes that create the data store(s). A job may include scheduling metadata which enables the process flow or user-supplied program to be executed in batch mode at a specified date and time.

Job Flow
is a user-defined diagram in the Job View of the Process Editor. It is composed of symbols, with connecting arrows and descriptive text, that illustrate the sequence in which Jobs and Events are executed. Job Flows are not required.

SAS/Warehouse Administrator allows you to create Job Flows, which define metadata for Job dependencies. However, the current release does not generate code for the Job Flows. To use them, you must write a Metadata API program that reads Job Flows and generates code for them. For details about writing metadata API programs, see *SAS/Warehouse Administrator Metadata API Reference*.

Job Group
is a simple grouping element for Process Editor Jobs, Events, and other Job Groups.

Load process (Load Step)
is a metadata record used to generate or retrieve a routine which puts data into a specified target object. After you define the metadata for a given data store, you must define a Load process which actually creates and loads the data store.

load time
is the date and time when a data value is loaded into a table.

logical model
is a model of a specific business process or concept from an end-user's perspective. A logical model identifies the subjects and relationships among data elements but does not describe the functional or physical characteristics of the data elements.

Mapping
is a metadata record used to generate or retrieve a routine which 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).

metadata
is a definition or description of data. Using the windows in SAS/Warehouse Administrator, you specify metadata which defines data sources, data stores, code libraries, and other warehouse resources. SAS/Warehouse Administrator then uses

this metadata to generate or retrieve the code which extracts, transforms, and loads the data into your warehouse.

There are two main kinds of metadata: physical metadata and business metadata. 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. 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.

metadata application program interface (API)
is a set of software tools that enable you to read or write SAS/Warehouse Administrator metadata without going through its user interface.

metadata repository
is a data store that contains an application's metadata. SAS/Warehouse Administrator stores its metadata in two SAS libraries: libref _MASTER (metadata for an Environment) and libref _DWMD (metadata for a Warehouse).

MOLAP
is multidimensional OLAP: online analytical processing performed on a multidimensional database, such as a SAS MDDB. SAS/Warehouse Administrator supports MOLAP with an OLAP Group of type *MOLAP*. Such a group is a grouping mechanism intended to contain only OLAP MDDBs. Multiple MDDBs can be contained in the group, but each MDDB generally represents an entire OLAP application.

MDDB
is a multidimensional data base. See OLAP MDDB.

NWAY crossing
is the most detailed type of crossing for an OLAP object. An NWAY crossing consists of all the assigned class columns.

ODD
is an operational data definition, a metadata record which provides access to data sources. The ODDs, in turn, are used as inputs to data stores in your warehouse.

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 that registers the location of a DBMS table with the help of a *SAS/ACCESS LIBNAME definition*. You can also define an ODD that extracts information from a data source, saves the results to a SAS table or view, then specifies the location of the extraction table or view. See also Data File and External File.

ODD Group
is a metadata record which specifies a group that is used to organize ODDs. It may also contain Information Marts.

OLAP
is online analytical processing, a kind of data analysis that is designed to answer "what happened and why" questions. OLAP enables you to easily and selectively extract and view data from different points of view. For example, a user can request that data be analyzed to display a report showing all of a company's products sold in Virginia in the month of September, 2000, then compare revenue figures with those for the same products in September, 1999, and then see a comparison of other product sales in Virginia in the same time period. SAS/Warehouse Administrator supports OLAP with its OLAP Groups, OLAP Tables, and OLAP MDDBs.

OLAP cube
is a metadata record that represents the logical relationships (dimensions and hierarchies) of the OLAP data so that you can run an OLAP report. For HOLAP, the

cube is associated with the OLAP Group and is registered in SAS/EIS software as associated with the proxy MDDB. The result is one OLAP Cube describing all the separate summary data stores. For MOLAP and ROLAP, there is normally one cube associated with each OLAP MDDB and OLAP Table.

OLAP Group
is a metadata record that specifies a group used to organize related summary data, which is stored in OLAP Tables and/or OLAP MDDBs. OLAP Group properties specify the logical structure of the summarized data. OLAP Groups replace the Summary Groups used in earlier releases.

OLAP MDDB
is a metadata record that specifies a SAS MDDB (multidimensional database). 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 replace the Summary MDDBs used in earlier releases.

OLAP Table
is 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. To load an OLAP Table, SAS/Warehouse Administrator generates code for the SUMMARY procedure, which summarizes data by computing descriptive statistics for columns across all rows or within groups of rows. OLAP Tables replace the Summary Tables used in earlier releases.

OLTP systems
are online transaction processing systems, the systems that an enterprise uses for processing business transactions on a day-to-day basis.

operational data
is data from an OLTP system. Operational data is a common source for the information that is extracted and loaded into a warehouse data store.

physical metadata
is a set of software instructions that define how an application element stores, moves, or transforms data. 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.

physical model
is the database design that identifies the structure and function of the database. A physical model is based on a logical model and is a technical specification that identifies schemas, tables, columns, indexes, and so on.

post-load process
is a routine that executes after a table is loaded. Post-load processes are specified on the Source Code tab of a load process window, which you display from the Process Editor.

process
is a routine that creates a warehouse data store, or that extracts data, transforms data, or loads data into a data store. In SAS/Warehouse Administrator, you define metadata records that are used to generate or retrieve the source code for processes. Mappings, User Exits, Data Transfers, Record Selectors, and Load Steps are all metadata records that generate or retrieve 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.

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 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.

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, although you may want to create one for documentation purposes.

Process Library
is a collection of registered routines that extract data, transform it, and/or load it into warehouse tables. For example, a Process Library might include routines that do a range-check on certain data values, or that eliminate duplicate values. As you use the Process Editor to define a step in the Process Flow for a particular table, you have the option of selecting a predefined routine from the Process Library, instead of defining your own process for that step.

proxy MDDB
See the discussion under HOLAP.

Record Selector process
is a metadata record used to generate or retrieve a routine which subsets data prior to loading it to a specified table. In the current release, a Record Selector can be used only to subset the source data specified in an ODD or in a Data File (which is an input to an ODD).

ROLAP
is relational OLAP: online analytical processing performed on a relational database, such as a SAS table or an ORACLE table. SAS/Warehouse Administrator supports ROLAP with an OLAP Group of type *ROLAP*. Such a group is a grouping mechanism intended to contain only OLAP Tables. Multiple OLAP Tables can be contained in the group, but each table generally represents an entire OLAP application.

SAS Component Language (SCL)
is a programming language used with SAS/AF and SAS/FSP software to develop interactive SAS applications. It was formerly known as SAS Screen Control Language.

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. SAS library definitions are included in the metadata records for data stores, processes, and Jobs in the current Environment.

SAS/ACCESS LIBNAME definition
is a special SAS library definition that can be used to extract source data in DBMS format or to create warehouse data stores in a DBMS. SAS/Warehouse Administrator uses a SAS/ACCESS 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 SAS/ACCESS 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.

SAS/Warehouse Administrator
is an application which provides a visual environment for managing data warehouses. Using the windows in this application, you specify the metadata which

defines data sources, data stores, code libraries, and other warehouse resources. SAS/ Warehouse Administrator then uses this metadata to generate or retrieve the code which extracts, transforms, and loads the data into your warehouse.

Scheduling Server definition
is a metadata record which specifies a scheduling server application (such as CRON under UNIX System V), a definition for the host where the scheduling server runs, directories where the scheduling server can send temporary files, the commands used to start SAS on the scheduling server host, and the default job-tracking option for jobs using this scheduling server definition.

star schema
is an arrangement of database tables in which a large fact table that has a composite, foreign key is joined to several dimension tables. Each dimension table has a single primary key.

statistic column
a required OLAP summary role, which is a numeric column for storing computed summary statistics...these are the measures you want to analyze. You must have at least one statistic column, with an unlimited maximum. The values for the input column (analysis column) are used to compute the output summary statistics, which then become the values for the statistic column in the summary data store.

For example, you could add a column named MINSALES, assign it as a statistic column using the MIN statistic, then define a Mapping process to compute the derived statistic from an analysis column like SALES to the statistic column MINSALES.

Each statistic column has a specific keyword associated with it that specifies which statistic to compute.

Subject
is a grouping element for data related to one topic within a Data Warehouse. For example, a Data Warehouse might have a Subject called *Products* (information related to products) or *Sales* (information related to sales). Each Subject may be composed of a number of different data collections: detail data, summary data, charts, reports, and graphs.

summary data
is information that is derived from the facts in a data base. It is data that has been summarized after extraction from a source.

summary data store
in SAS/Warehouse Administrator, OLAP Tables or OLAP MDDBs that store crossings and statistic columns. As each crossing is accumulated, all records of the input data are sorted into groups by the values of the class columns included. Each group represents a specific subpopulation. Within each group, the summary statistics are calculated on all input records and the statistics are stored in statistic columns with the same name as the analysis columns they were derived from. A single record is written to the crossing for each subpopulation.

Summary Group
has been replaced by OLAP Group. These are provided for compatibility with earlier releases of SAS/Warehouse Administrator.

Summary MDDB (Multidimensional Database)
has been replaced by OLAP MDDB. These are provided for compatibility with earlier releases of SAS/Warehouse Administrator.

Summary Table
has been replaced by OLAP Table. These are provided for compatibility with earlier releases of SAS/Warehouse Administrator.

User Exit process
is a metadata record used to retrieve a user-written routine. The routine must be stored 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.

Warehouse Environment
See Data Warehouse Environment.

Index

Your Turn