SAS® 9.1 OLAP Server
Administrator's Guide

The Power to Know®

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2004. *SAS® 9.1 OLAP Server: Administrator's Guide.* Cary, NC: SAS Institute Inc.

SAS® 9.1 OLAP Server: Administrator's Guide

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

1st printing, January 2004

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

Contents

What's New

Overview

The SAS OLAP Server enables users to develop and deploy scalable Online Analytical Processing (OLAP) applications. In addition, automated data loading and cube building is available through the use of a new administration interface called the SAS OLAP Cube Studio, which was developed using Java technology.

OLAP queries are performed using the Multidimensional Expressions (MDX) query language in client applications that are connected to the OLAP Server by using

- the SQL Pass-Through Facility for OLAP, which is designed to process MDX queries within the PROC SQL environment.
- open access technologies such as OLE DB for OLAP, ADO MD, and Java.

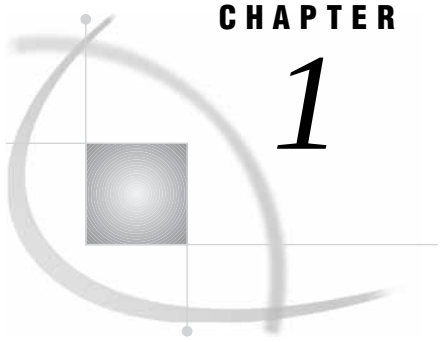
Note: This section describes the features of the SAS OLAP Server that are new or enhanced since SAS 8.2. △

Details

- There are two new tools for data loading and cube building:
  - The OLAP procedure, in addition to cube building, includes options for handling ragged hierarchies, defining global calculated members and named sets, assigning properties to levels, and optimizing cube creation and query performance. It also supports multiple hierarchies and drill-through tables.
  - The SAS OLAP Cube Studio is an alternative Java interface to the OLAP procedure. This interface is also integrated with SAS ETL Studio.
- Server performance is recorded and analyzed by using the Application Response Measurement (ARM) system.
- The new multi-threaded data storage and server functionality provide faster cube performance. The data can be stored in a multidimensional form (MOLAP) or in a form that includes existing aggregations from presummarized data sources.
- The metadata structure is improved, and metadata is stored with the cube.

- Caching and logging can be enabled or disabled.
- Support for ad hoc calculations and time dimensions is improved.
- An SQL Pass-Through Facility for OLAP is available in SAS for use in querying cubes.
- Aggregations can be added to or deleted from existing cubes.

Note: Version 8 of the SAS OLAP Server can be used with SAS 9. For help, see "V8 SAS OLAP Server" in SAS System Help and Documentation. △



CHAPTER

1 OLAP Introduction and Overview

What Is OLAP?

Online Analytical Processnding (OLAP) is a technology that is used to create decision support software. OLAP enables application users to quickly analyze information that has been summarized into multidimensional views and hierarchies. By summarizing predicted queries into multidimensional views prior to run time, OLAP tools provide the benefit of increased performance over traditional database access tools. Most of the resource-intensive calculation that is required to summarize the data is done before a query is submitted.

Data Storage and Access

Decision makers are asked to make timely and accurate decisions that are based on the past performance and behavior of an organization as well as on future trends and directives. To make effective business decisions, business analysts must have access to the data that their company generates and responds to. This access must include timely queries, summaries, and reviews of numerous levels and combinations of large, recurrent amounts of data. The information that business analysts review determines the quality of their decisions.

Organizations usually have databases and data stores that maintain repeated and frequent business transaction data. This provides simple yet detailed storage and retrieval of specific data events. However, these data storage systems are not well suited for analytical summaries and queries that are typically generated by decision

makers. For decision makers to reveal hidden trends, inconsistencies, and risks in a business, they must be able to maintain a certain degree of momentum when querying the data. An answer to one question usually leads to additional questions and review of the data. Simple data stores do not successfully support this type of querying.

A second type of storage, the data warehouse, is better suited for this. Data is maintained and organized so that complicated queries and summaries can be run. OLAP further organizes and summarizes specific categories and subsets of data from the data warehouse. This results in a robust and detailed level of data storage with efficient and fast query returns. SAS OLAP cubes can be built from either partially or completely denormalized data warehouse tables. Stored, precalculated summarizations called *aggregations*, can be added to the cube to improve cube access performance. Aggregations can either be pre-built relational tables, or you can let the cube create its own optimized aggregates.

Benefits of OLAP

The ability to have coherent and relevant information is the reason OLAP has gained in popularity. OLAP systems help reveal evasive inconsistencies and trends in data that might not have been seen before. OLAP users can intuitively search data that has been consolidated and summarized within the OLAP structure. In addition, OLAP tools allow for tasks such as sales forecasting, asset analysis, resource planning, budgeting, and risk assessment. OLAP systems also provide the following benefits:

- fast access, calculations, and summaries of an organization's data
- support for multiple user access and multiple queries
- the ability to handle multiple hierarchies and levels of data
- the ability to pre-summarize and consolidate data for faster query and reporting functions
- the ability to expand the number of dimensions and levels of data as a business grows.

To fully understand the benefits of OLAP and the details of its effective implementation, it helps to examine the technology from two perspectives—first, from that of the users and second, from that of the information technology (IT) administrators who are responsible for OLAP implementation. The users, typically business analysts and analysts, expect the data to be organized according to categories that reflect the way in which they think about the enterprise. For IT administrators, OLAP can present a long list of technical issues, including these concerns:

- storage requirements and associated costs
- client and server capabilities
- maintenance activities such as update and backup
- performance considerations such as the amount of time that is required to build a multidimensional model
- the ability of the OLAP solution to integrate with current or planned data warehouse strategies and architectures.

What Is a Cube?

One of the advantages of OLAP is how data and its relationships are stored and accessed. OLAP systems house data in structures that are readily available for detailed queries and analytics. Cubes are central to the OLAP storage process.

A *cube* is a set of data that is organized and structured in a hierarchical, multidimensional arrangement. The cube is usually derived from a subset of a data warehouse. Unlike relational databases that use two-dimensional data structures (often in the form of columns and rows in a spreadsheet), OLAP cubes are logical, multidimensional models that can have numerous dimensions and levels of data. Also, an organization typically has different cubes for different types of data.

One of the challenges of OLAP cube data storage and retrieval is the growth of data and how that growth affects the number of dimensions and levels in a cube hierarchy. As the number of dimensions increases over time, so does the number of data cells on an exponential scale. To maintain the efficiency and speed of the OLAP queries, the cube data is often presummarized into various consolidations and subtotals (aggregations).

Note: The SAS OLAP Server term *cube* is synonymous with the terms *hyper-cube* and *multi-cube*. △

Understanding the Cube Structure

OLAP cubes organize data in a hierarchical arrangement. Data is structured according to dimensions and measures.

Dimensions group the data along natural categories. (Examples of dimensions are Time, Products, Organization). Typically, dimensions offer different levels of grouping (for example, the Time dimension can be grouped by Years, Months, Days, etc.). *Levels* are organized into one or more hierarchies, typically from a coarse-grained level (for example, Year) down to the most detailed one (for example, Day). The individual category values (for example, 2002 or 21Jan2002) are called *members*.

Measures are the data values that are summarized and analyzed. Examples of measures are sales figures or operational costs. The data for measures is located in *cells*. Cells are the intersection of one member for every dimension.

Presummarized data in a cube is stored in aggregations. Aggregations are the basis for fast response to data queries in OLAP applications. An aggregation is possible at each intersection of a level of one or more dimensions. The selection of aggregations to presummarize is one of the major factors that determine query response time and cube size.

What Is SAS OLAP Server?

SAS OLAP Server is a scalable server that provides multi-user access to the data that is stored in SAS OLAP cubes.

Processing data by using a multi-threaded kernel enables you to take advantage of your server's parallel processing abilities. SAS OLAP Server accepts data queries in the industry-standard MDX query language, which opens it up to a variety of clients. Other features include

- the SAS OLAP Cube Studio user interface, which is an alternative Java interface, for building and maintaining cubes
- PROC OLAP for programmatically building and maintaining cubes
- server management by using SAS Management Console
- support for processing external aggregates
- support for OLE DB for OLAP.

Note: OLAP queries are performed by using the Multidimensional Expressions (MDX) query language in client applications that are connected to the OLAP server by

using OLE DB for OLAP (an extension of OLE DB that is used by COM-based clients), or through a similarly designed Java interface. △

About Cube Metadata Storage

The SAS Metadata Server stores the metadata that defines the cubes. It is a multi-user server that enables users to manage metadata in one or more metadata repositories by using the SAS Open Metadata Interface.

Note: The SAS Open Metadata Interface is an object-oriented application programming interface (API) that interacts with the SAS Metadata Server. SAS OLAP Cube Studio is an example of an application that is compliant with the SAS Open Metadata Interface. △

The SAS Metadata Server uses the Integrated Object Model (IOM) that is provided by SAS Integration Technologies. IOM provides distributed object interfaces to Base SAS software features. It enables you to use industry-standard languages, programming tools, and communication protocols to develop client programs that access these services on IOM servers.

In the SAS OLAP Server, all relevant structural information is contained within the cube and most of it is also replicated within the SAS Open Metadata Architecture. This is done so you can

- disassociate the cube definition process from cube creation, which enables you to create a cube by using its stored definition
- define and enforce security at the SAS Open Metadata Architecture level
- manage and control the data source in the centralized Open Metadata Architecture repository.

Documentation about the SAS Open Metadata Architecture is available at **http://support.sas.com.**

Why You Should Use Cubes

SAS cubes are designed to offer efficient data storage, fast data access, easy data maintenance, and flexibility in data management. The following sections explore cubes and multidimensional storage.

Cube Usage and Storage Space Reduction

While cubes are the format of choice to guarantee fast query response times against your data warehouse, SAS OLAP cubes are also often a very space efficient choice for data storage. In many cases, a basic cube without additional aggregations can be smaller than the input data because the process of creating the cube consolidates records. SAS OLAP cubes use the hierarchy information for efficient aggregations storage. SAS OLAP cubes also deal efficiently with data sparsity by using virtual placeholders for empty cells. This removes the need for any physical representation of empty cells. A good rule of thumb is, the larger your input data, the greater the storage gain by loading data into a cube.

Multi-Threading Capabilities

Loading data into cubes and executing queries against the cube take advantage of the multi-threading capabilities of your server machine. Aggregations are created in parallel at cube build time. The creation of individual aggregations takes advantage of the Parallel Group-By capabilities of SAS' data engine. At query execution, the multi-threading capabilities of your server machine are fully used to concurrently serve queries by multiple users. Both query evaluation and data access are executed in parallel. To further increase query performance and reduce disk access, you can allocate additional memory on your server to be used for an in-memory aggregation cache.

Easy Setup and Maintenance

A cube is the physical representation of your logical dimensional model. The tools that are provided to update and maintain the cube reflect the multidimensional model, which makes both setup and maintenance of your cube as intuitive as possible. SAS' thin-client, Web-based administrator interface, SAS Management Console, enables you to set up and manage OLAP servers. SAS OLAP Cube Studio provides the workspace and cube designer tools that you need to create and maintain cubes. You can also use the SAS OLAP procedure to create and maintain cubes in a batch environment.

Data Management: Choosing Your Own Tool

If you create your own aggregations by using data management tools such as SQL, PROC SUMMARY, or the tools of your preferred relational database management system (RDBMS), then you can link those aggregations to your cubes without replicating the data within the cube. Any queries against those aggregations are executed by the appropriate SQL engine, and take advantage of any capabilities that engine might have. This allows you the flexibility to use the data management tools of your choice. It also allows you to distribute your data for your cube aggregations across multiple database systems, servers, and platforms. If you choose to let the cube builder create the aggregations, then you can control where to store the data and index files for each aggregation.

Analyzing Your Data

Data Preparation and Dimension Design

The goal of an OLAP system is to have data that is organized, available, and presented as relevant information to decision makers. OLAP cubes are based on data from data warehouses. A data warehouse consists of data that is extracted from transactional systems at regular intervals. The extraction process works very closely with data quality control, making sure that the data is complete and accurate. Extensive data cleansing (which includes eliminating variant spellings of names) can be part of this task.

Building a data warehouse also implies transforming data that is optimized for transactional processing into data that is optimized for user-driven analysis. Part of that process is grouping facts and attributes into entities that correspond to the users'

view of the organization. These groupings are known as dimensions. For related information, see the SAS online documentation for SAS ETL Studio.

An established technique for implementing a dimensional model is to create star join schemas that are based on the data. SAS OLAP cubes can be loaded from star schemas, or from further denormalized tables or views that include some or all dimensions in the fact table.

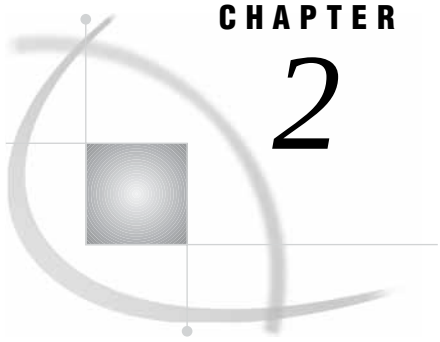
Aggregation Design

Efficient drilling or traversing of the cube data is a key factor in flexible and quick decision making and analysis. In order to maintain speed and consistency in reporting, data is usually precalculated or aggregated. An important factor in query performance is good aggregation design, which includes decisions about total storage space, available build time, storage location, and storage format.

When planning your data storage and design, it is helpful to approximate the size of aggregations. A basis for estimating aggregation size is the number of distinct values in a dimension level, otherwise known as cardinality. The other factor that determines aggregations size is density. Density is a measure of how many members of each dimension in an aggregation occur in combination with the members of the other dimensions (for example, there might not be sales of a specific product on a specific date). The total cube size as well as the resources that are available for the cube build process determine the build time that is needed. It is also important to note that build time should not exceed the cube update interval.

Aggregation size and available hardware influence your choices for aggregation partitioning. You can separate aggregations into multiple files. A reduced file size might accelerate OLAP server access time, particularly if multiple processors are available for multi-threaded processing. You can use preaggregated summary tables, the cube's own efficient aggregation storage, or a combination of both. Using indexes on either storage type might increase query performance, while also increasing storage space and build time.

After an initial aggregation design is chosen, subsequent cube builds enable you to optimize the cube's performance and size by adding or removing aggregations. You can analyze the OLAP users' behavior by using Application Response Measurement (ARM) logs, showing which aggregations are needed most and would be the most efficient.



CHAPTER

2

Installing and Administering SAS OLAP Server

Installing and Configuring SAS OLAP Server

To create SAS OLAP cubes, you must complete several installation and configuration steps. For our purposes, you create a cube using one of the following methods:

- SAS OLAP Cube Studio
- PROC OLAP commands in a SAS session.

To create a cube with either method, complete these steps:

1. Define a metadata repository.
2. Invoke a metadata server.
3. Define a basetable library.
4. Set up an OLAP schema definition in the metadata repository.

And specifically when you create a cube with SAS OLAP Cube Studio, complete these steps:

1. Invoke a SAS Workspace Server and set configuration parameters.
2. Create tables that are used to build cubes in the SAS Workspace Server.

Two servers are necessary to build OLAP cubes—a SAS Metadata Server and SAS Workspace Server. A third server, the SAS OLAP Server, is necessary when you query OLAP cubes. The installation and setup of the SAS Metadata Server, the SAS Workspace Server, and SAS OLAP Cube Studio combine the installation and configuration of various SAS components that set parameters, directories, file paths and data storage locations. The setup includes the following tasks:

- Installing the required components for SAS Management Console, SAS OLAP Cube Studio, and SAS OLAP Server
- Configuring the required directories, permissions, repositories, servers, spawners, libraries, and batch files.

Note: SAS OLAP Server is licensed to start and access a SAS Workspace Server for use with SAS OLAP Cube Studio. Users who have permission to create and maintian cubes in SAS OLAP Cube Studio can access SAS workspace servers.

If you do not have a separate, unrestricted license to use SAS ETL and SAS Integration Technologies on the server that SAS OLAP Server is deployed to, then you

must restrict use of SAS workspace servers to SAS OLAP Cube Studio. This is to avoid restricted license violations of SAS workspace servers. △

Product Installations

The following components must be installed to configure the OLAP environment:

- JRE 1.4.1
- SAS 9.1
- SAS Management Console 9.1
- SAS Metadata Server 9.1
- SAS OLAP Cube Studio 9.1.

SAS Configuration Wizard

The OLAP Server installation is part of the overall installation and configuration of SAS servers. The SAS Configuration Wizard is available for automated configuration of SAS servers and handles many of the server configuration steps for you. After you have installed the necessary product components that are listed under Product Installations, the SAS Configuration Wizard can perform various automated configuration tasks. After the wizard completes the automated tasks, an HTML document is generated that lists the remaining manual configuration steps that you need to perform. If you choose not to use the configuration wizard to set up your SAS Metadata and SAS OLAP servers, then you can use the steps below to configure and set up these servers.

Note: For more information about the SAS Configuration Wizard, see the *SAS Intelligence Architecture: Planning and Administration Guide* at **`http://support.sas.com`**. △

Configuring and Setting Up Open Metadata Architecture

Configuring Open Metadata Architecture

The SAS Metadata Server and the SAS Open Metadata Architecture are part of the SAS 9.1 product installation. The SAS Open Metadata Architecture enables you to create metadata repositories and set up metadata servers. To configure the Open Metadata Architecture, you will need to set up default directories, assign permissions, and generate a configuration XML file. To create the necessary directories on Windows, complete these steps:

1. On the C: drive, select

   File ▶ New ▶ Folder

   to create a folder. Name the folder *sas server config*. This is a user defined name.
2. In the *sas server config* folder, create a *Lev1* folder.
3. In the *Lev1* folder, create a *SASMain* folder.
4. In the *SASMain* folder, create a *MetadataServer* folder. You should now have a folder path of:

   c: ▶ sas server config ▶ Lev1 ▶ SASMain ▶ MetadataServer
5. Open the *MetadataServer* folder and create two new folders. Select

File ► New ► Folder

twice. Name the first folder *rposmgr*. Name the second folder *MetadataRepositories*. In the *MetadataRepositories* folder, create a folder called *Foundation*.

Note: If you create directories for your repositories as subdirectories of the Metadata Server directory, as shown above, then the repositories will inherit file and directory access permissions from the server directory. △

Setting Up Directory and File Access Permissions for Open Metadata Architecture

To set up or maintain a repository manager and repository, you must have access to the repository manager and repository directories. In this step, you specify permissions for the *MetadataServer* server directory that you created. It is assumed that the *rposmgr* and *MetadataRepositories\Foundation* directories are nested within the Metadata Server folder so they can inherit permissions.

For Windows operating systems, you right-click the Metadata Server folder and select the **Properties** option. From here you set archiving conditions and permissions. For UNIX and MVS systems, see the operating system commands that are appropriate to your host environment to set directory and file protections. Steps in the setup process vary depending on your operating system.

Note: For more information, see the *SAS Metadata Server: Setup Guide* at **http://support.sas.com**. △

Setting Up System Access Permissions for Windows Operating Environments

It is necessary to set up system access permissions to accommodate the SAS Metadata Server. Depending on your Windows operating system, select the **User Rights** or **User Rights Assignment** option from the Windows administrative tools folder. Select and add these policy settings for access permissions:

- Act as part of the operating system (Windows NT and Windows 2000 only)
- Log on as a batch job. Assign the *Everyone* identity as server accessor. This enables all users and applications to access the metadata server.

  Note: As an alternative, you might consider defining a SAS Users group and assign the Log on as batch job user right to this group instead. △

Note: The process for setting user rights depends on your Windows operating environment. For more information, see the *SAS Metadata Server: Setup Guide* at **http://support.sas.com**. △

Setting Up System Access Permissions for UNIX Operating Environments

The SAS Metadata Server requires that the SASPERM and SASAUTH files in the **!SASROOT/utilities/bin** directory be setuid and owned by root. These permissions are typically set at SAS installation by using the setup utility. You might want to verify that the appropriate permissions are set. If they are not, change setuid to root using one of the following methods:

Method 1 – Using SAS Setup

1 Log in to the root account.

```
$ su root
```

2 Run SAS Setup from **!SASROOT/sassetup**.
3 Select **Run Setup Utilities** from the SAS Setup Primary Menu.
4 Select **Perform SAS System Configuration**.
5 Select **Configure User Authorization**.

Method 2 At a UNIX prompt, type the following commands:

```
$ su root
# cd !SASROOT/utilities/bin
# chown root sasauth sasperm sasrun
# chmod 4755 sasauth sasperm sasrun
# exit
```

Note: For more information, see the *SAS Metadata Server: Setup Guide* at **http://support.sas.com**. △

Creating the omaconfig.xml File

The SAS Metadata Server supports an omaconfig.xml file to enable you to specify alternate settings for server configuration options. The omaconfig.xml file is an optional file that you create in the current working directory to identify the configuration options that you want to change and their new settings. To create an omaconfig.xml file, complete these steps:

1 In the server directory, use your favorite text editor to create a file named "omaconfig.xml".
2 In the file, insert XML elements that represent the server configuration options whose defaults you want to change. Configuration options whose elements are omitted from the omaconfig.xml file retain their default values.

Note: For SAS OLAP purposes, the current working directory is **SASMain\MetadataServer** within the directory structure **c:\sas server config\Lev1\SASMain\MetadataServer**. This directory contains the server startup scripts, the trusted user and administrative user text files, and the Open Metadata configuration files. △

Note: For more information, see the *SAS Metadata Server: Setup Guide* at **http://support.sas.com.** △

Configuring Special Users

The metadata server supports the use of text files to register user IDs that are given special privileges on the server. The server recognizes two classes of special users:

Administrative user — is a user ID that has been given permission to perform administrative tasks on the server. These user IDs are identified in an *adminUsers.txt* file.

Trusted user — is a user ID that acquires credentials on behalf of other users in a multi-tier server environment. The trusted user ID functionality is typically required by applications. These user IDs are identified in a *trustedUsers.txt* file.

The adminUsers.txt and trustedUsers.txt files are created in the server directory and read when the server is started. For further information, see "Configuring Special Users" and "Using the Authorization Facility" in the *SAS Metadata Server: Setup Guide* at **http://support.sas.com**.

Starting the Metadata Server—Creating an Executable File

After all the necessary directories and configuration files have been created, you must start the server in a non-interactive SAS session. The server can be started by passing the following basic parameters in the start command. For a complete listing of recommended and optional parameters see the *SAS Metadata Server: Setup Guide* at **http://support.sas.com**.

Windows:

```
"where_your_sas_is_installed\sas.exe" -nosplash -noterminal -sasuser sasusrms
-rsasuser -log "logs\sasoms.log" -logparms "write=immediate" -linesize max
-pagesize max  -memsize max -objectserver -objectserverparms "protocol=bridge
port=8561 classfactory=2887E7D7-4780-11D4-879F-00C04F38F0DB"
```

where_your_sas_is_installed is the path to the directory where SAS 9.1 is installed at your site, typically C:\program files\sas\sas system\9.1 and *port=8561* is the default port to which the server listens for client requests. Edit the port value to a number that uniquely identifies the metadata server at your site. For convenience, you might want to store the command in a **startsrv.bat** file that you can execute to start the server. The **startsrv.bat** file is stored in the MetadataServer directory and should be executed from that directory.

UNIX:

```
 [sas-cmd] -noterminal -sasuser sasusrms -rsasuser -log log/sasms.log
-logparm "write=immediate" -linesize max -pagesize max -memsize max
-objectserver -objectserverparms "protocol=bridge port=8561
classfactory=2887E7D7-4780-11D4-879F-00C04F38F0DB"
```

```
[sas-cmd]
```

is the command that is used to invoke SAS at your site, and *port=8561* identifies the port to which the server listens for client requests. Edit the port value to a number that uniquely identifies the metadata server at your site. For convenience, you might want to store the command in a shell script that you can execute.

OS/390

For information about starting the SAS Metadata Server in the OS/390 environment, see the *SAS Metadata Server: Setup Guide* at **http://support.sas.com**.

Note: The SAS Metadata Server starts in a default secured mode, which means that only authenticated user IDs have access to the server. △

Note: To stop the metadata server, navigate to the **Metadata Manager/Active Server** function in SAS Management Console. You must right-click the **Active Server,** and then select the **Stop** function.

You can also execute the PROC METAOPERATE statement:

```
PROC METAOPERATE
 SERVER="localhost"
 PORT=1111
 USERID="myuserid"
 PASSWORD="mypassword"
 PROTOCOL=BRIDGE
```

```
 ACTION=STOP;
RUN;
```

For further information, see "Starting and Stopping the Server" in the *SAS Metadata Server: Setup Guide* at **http://support.sas.com**. △

Note: For further information about SAS Metadata Server parameters that are used in server startup scripts, see "Starting the SAS Metadata Server" in the *SAS Metadata Server: Setup Guide* at **http://support.sas.com**. △

Starting the Metadata Server as a Windows Service

SAS uses the SAS Service Configuration Utility (SSCU) to configure services. To set up the metadata server to run as a service, complete these steps:

1. Create your omaconfig.xml, adminUsers.txt, and trustedUsers.txt files.
2. Make sure that the domain-qualified user ID that you will use to run the server is in the adminUsers.txt file.
3. On the Windows Start menu, select

   Programs ► your-SAS-System-folder ► SAS 9.1 Utilities ► SAS Service Configuration Utility
4. Enter the following information in the SAS Service Configuration Utility (SSCU):

   □ On the Install tabbed page:

   Service Name — is a name to register the server to Windows. The service name is also the name that is used when a net start or a net stop command is issued. The recommended service name for the metadata server is SASOMS.

   Display Name — is the name of the service that is displayed to user-interface applications. The recommended display name is SAS Metadata Server.

   Start Type — specifies how the server is started. Select **Automatic**.

   Service Path — specifies the command to invoke the metadata server. Type the batch SAS Metadata Server start command shown in the previous section. Here is an example:

   ```
   C:\Program Files\SAS\SAS 9.1\sas.exe" -nosplash -noterminal
   -sasuser sasusrms -rsasuser -log "sasoms.log" -logparm
   "write=immediate"  -linesize max -pagesize max -memsize max
   -objectserver  -objectserverparms "protocol=bridge port=8561
   classfactory=2887E7D7-4780-11D4-879F-00C04F38F0DB"
   ```

   Working Path — is the pathname of the directory where the metadata server is defined. This is the directory that is created for the server, as described in "Creating Directories for the Metadata Server," "Repository Manager," and "Repositories." In this example, the working path is c:\omaserver.

   Description — is an optional description of the service. The description appears in the Windows Services window Details pane.

   □ On the Account tabbed page:

   This Account — Type the user ID under which the metadata server will run. The user ID that you enter here should match the user ID that you specified in the adminUsers.txt file earlier.

- □ Return to the Install tabbed page, and click **Install**.

5 Confirm that your newly defined service is listed among other services running on your machine and start it.

a Select

Start ▶ Settings ▶ Control Panel

b Double-click **Administrative Tools**.
c Double-click **Services**.
d Scroll down and verify that SAS Metadata Server appears in the list.
e Right-click **SAS Metadata Server**, and then select **Start**.

If the metadata server does not start successfully, check the SAS log for messages indicating reasons for the failure. For further information, see the *SAS Metadata Server Trouble-Shooting Q&A* at **http://support.sas.com**.

Note: For further information about starting a server as a Windows service, see the *SAS Companion for Windows*. △

Setting Up and Configuring SAS Management Console and SAS OLAP Cube Studio

SAS Management Console—Setting Up Repositories

In SAS Management Console, create the metadata profile and the required repositories with the Metadata Manager. When you set up the metadata profile, add two different types of repositories:

- □ the Foundation (parent) repository
- □ any required Custom (child) repositories.

In SAS Management Console, under **Metadata Manager**, select and right-click the **Active Server**. Select **Add Repository**. This loads the Select Repository Type window.

- □ Add the Foundation (parent) repository:
  - □ For the parent repository, select **Foundation**.
  - □ Define the engine and path information, and then select **Finish** in the Current Settings window.
  - □ Pause the server when prompted.

  Note: In the Definition of Data Source window, do not select the change management option for the parent or Foundation repository. △
- □ Add any needed Custom (child) repository(s):
  - □ For the child repositories, select **Custom**.
  - □ In the Define Repository Dependencies window, select the foundation repository that you defined earlier as the repository that the child repositories depend on and receive user groups and permissions from.
  - □ Check the box for **Create direct dependencies to all parents of chosen directories**.
  - □ Select **Finish** in the Current Settings window.
  - □ Pause the servers when prompted.

Note: Resource template definitions are automatically added to the foundation repository when the repository is created. △

Note: See *SAS Management Console Help* for further information. △

SAS Management Console—Setting Up a SAS Workspace Server

You need to start a SAS server and define a SAS Workspace Server in the metadata repository if any of the following conditions are true:

- Your tables are not already registered in the metadata repository.
- You plan to create the physical cube in addition to registering its metadata with SAS OLAP Cube Studio.
- You want to use the Source Designer to define source tables for cubes.
- You want to use SAS OLAP Cube Studio to generate a cube.

Complete these steps to create the required SAS workspace servers:

1. Verify that the correct resource templates are loaded. You can check this by looking in these folders:

   Metadata Manager ▶ Resource Templates

2. In SAS Management Console, **`Metadata Manager`**, make sure that the selected repository is the repository you created as the foundation (parent) repository.
3. In SAS Management Console, select the **`Server Manager`** function. Right-click, and then select the **`New Server`** option.
4. In the New Server Wizard window, select the **`SAS Application Server`** option, and then define the new server.
5. Select the **`Workspace Server`** type option (the type of server that you are creating).
6. Select the **`Custom`** configuration setting method for the workspace server.
7. Enter the server properties, connection type (Bridge or COM), and connection properties.

   Note: For information on when to use the IOM Bridge versus the COM connection see "Choosing a Server Configuration" in the *SAS Integration Technologies Administrator's Guide* at **`http://support.sas.com`**. △

8. Repeat this process for each workspace server that you define.
9. You can now create an object spawner in SAS Management Console to start the workspace server.

For further information about COM and IOM Bridge connections and setup instructions, see "Setting Up a Server with a COM/DCOM Connection: Introduction" and "Setting Up an IOM Bridge Connection: Introduction" in the *SAS Integration Technologies Administrator's Guide* at the Enterprise Integration Community at **`http://support.sas.com.`**

Note: Specifically, if you are setting up a COM connection, you must complete the steps outlined in "Summary of Setup Steps (COM/DCOM) — Separate Client and Server machine" in the *SAS Integration Technologies Administrator's Guide* at the Enterprise Integration Community at **`http://support.sas.com.`** △

Using an Object Spawner to Control SAS Workspace Servers

The object spawner helps conserve resources. You are required to have an object spawner to control the workspace servers.

1. In SAS Management Console, select the correct metadata repository.
2. Select **Server Manager**, then right-click and select **New Server**.
3. In the New Server Wizard, select

   SAS Servers ▶ Spawner ▶ Object Spawner

4. Name the object spawner that you are creating and define the server and initialization properties.
5. Select the workspace servers as the servers that the object spawner will control.
6. Select the connection type and enter the connection properties.
7. Select **Finish** on the final screen.
8. When you are finished, you will see the spawner that you created listed under the Server Manager.

You can now create an executable file (batch) to run the object spawner. Create the file in the SAS 9.1. directory. Here is an example:

```
objspawn ---omrconfigFile omrconfig.xml
```

The object spawner requires that certain system access permissions be set. In addition to the permissions detailed in "Setting Up System Access Permissions for Windows Operating Environments" on page 10, you must add the following policy settings.

- Increase Quotas (This is labeled "Adjust memory quotas for a process" on Windows XP)
- Replace a process level token

Note: For further information about starting an object spawner, see *SAS Integration Technologies* at the Enterprise Integration Community at **http://support.sas.com.** △

Setting Memory Usage Options for a SAS Workspace Server

When you set up a SAS Workspace Server with the New Server Wizard, you define SAS system options that specify memory usage and cube loading. For a MOLAP cube that loads from a single input table, the SAS OLAP cube loading process occurs in four separate stages that overlap at certain points during execution. During cube loading, one of the stages is typically the dominant consumer of the system's resources. This applies to the following stages:

- Data summarization
- Metadata creation
- NWAY disking (the writing of the data values in the NWAY to disk).
- Subaggregation creation.

There are two factors that affect memory usage during cube loading:

- the amount of physical memory and virtual memory that is installed on the machine
- the value of the SAS options MEMSIZE, REALMEMSIZE, and SUMSIZE.

If any of these options are set to a value that is greater than zero, then they override the actual physical memory and virtual memory values. As a result, it is recommended to minimally specify the MEMSIZE and REALMEMSIZE options to the actual amount of installed memory and virtual memory. If they are not specified, then the cube loader attempts to allocate memory.

To specify these options in the New Server Wizard window of SAS Management Console, complete these steps:

1 Select the **Custom** configuration setting method for the new workspace server.
2 In the Command field, enter the MEMSIZE and REALMEMSIZE options as needed at the end of the SAS command.

Note: An alternative is to store these options in a configuration file and reference that file with the CONFIG= option on the command line. For details, see the CONFIG system option in the *SAS Companion for Windows*. △

You can also specify these options in SAS OLAP Cube Studio. In the

Cube Designer ▶ Advanced Cube Options

window, enter these options as needed in the Submit SAS Code field. Use the OPTIONS statement to specify the memory options. For more information, see the OPTIONS statement in *SAS Language Reference: Dictionary*.

Creating a New Library Definition for Source Data Tables

You can create a new library definition for your source data tables after you start a SAS server and define a SAS Workspace Server in a SAS Metadata Repository. You must create a library definition if any of the following conditions are true:

- You have not already defined your tables in the currently active metadata repository.
- You plan to create the physical cube in addition to registering its metadata.
- You want to manually add, modify, or drop specific aggregations.

You create new library definitions by using the New Library Wizard, which is available from SAS OLAP Cube Studio, SAS ETL Studio, and SAS Management Console. The following steps explain how to use SAS OLAP Cube Studio to launch the wizard and define a new library:

1 Select

Tools ▶ Source Designer

Note: The Source Designer is also available from the Cube Designer. △

2 Select the SAS source type, and then click **Next** to continue.

Note: If you have not provided your user ID and password for the selected server, then you are prompted to log in. △

3 In the Select a SAS Library window, click **New** to launch the New Library Wizard.
4 Enter the library name. You can also enter an optional description. Click **Next**.
5 Enter the libref name, the engine type (BASE is the default), and the path specification. Follow these guidelines:
   - The libref is a short name (or alias) for the complete physical name of a SAS library (for example, sasuser).
   - The path specifies the physical location of the tables contained in the library that you are defining. Select an existing path from the box or click **New** to enter a new path.
   - Click the **Advanced Options** button to set options for any host such as file encoding, as well as host-specific options. Click **OK** to close the dialog box and return to the New Library Wizard. Click **Next.**
6 Select your defined SAS Workspace Server. You assign the library to this server. Click **Next**.

Note: The logical server name is used to identify a group of machines that are assigned to a particular server definition. However, only one machine per logical server is supported for SAS 9.1. △

7 Click **Finish** to complete the new library definition.

Note: The SAS Metadata Server allows duplicate librefs to be defined in the metadata. To ensure that the correct SAS library definition is found on the metadata server, you assign the libref by using the LIBNAME statement for the metadata engine before you submit the PROC OLAP code. Otherwise, PROC OLAP selects the first library definition that it finds with your specified libref, and it will associate your cube metadata with that definition. The selected library definition might or might not contain a description of the SAS data set that was used to build your cube. For more information about using the LIBNAME statement for the metadata engine, see "Statements" in *SAS Language Reference: Dictionary*. △

Storage Location Requirements for Cube Metadata and Related Objects

The metadata objects that describe the cube and the cube's associated libraries and source tables must be stored in the same repository, or the metadata that describes the cube must be in a custom repository that is dependent on the repository that contains the library and table objects. Otherwise, you are not be able to create the cube. In addition, the library and table objects that are referenced by a cube must always be in the same repository. The following options illustrate these conditions:

- The library, table, and cube objects can be in a Foundation repository.
- The library, table, and cube objects can be in Project A, which is dependent on the Foundation repository.
- The library and table objects can be in the Foundation repository, and the cube object can be in Project A.
- The cube object cannot be in the Foundation repository, and the library and table objects cannot be in Project A.
- The table object cannot be in the Foundation repository, and the library and cube objects cannot be in Project A.
- The library object cannot be in the Foundation repository, and the table and cube objects cannot be in Project A.

Using the SAS Workspace Server to Define Tables (That Are Used to Build Cubes)

Here is a list of the tables that are used to define a cube:

- Detail tables (unsummarized data)
- Fact tables and dimension tables (for cubes that are based on star schemas)
- Aggregation tables (fully summarized external tables)
- Drill-through tables (views maintained by the user that represent all of the data that is used to define a cube).

You must define the tables that are used to build cubes if any of the following conditions are true:

- You are not using SAS ETL Studio to load table definitions.
- You have not already defined your tables in the active metadata repository.

□ You want to manually add, modify, or drop specific aggregations.

Before completing these steps, a defined SAS Workspace Server must be running and you must have defined libraries for it. Complete these steps in SAS OLAP Cube Studio:

1 Select

Tools ▶ Source Designer

Note: The Source Designer is also available from the **Define Table** button, which you can select from the Cube Designer Input and Drill-Through windows. △

2 Select the SAS source type, and then click **Next**.

Note: If you have not provided your user ID and password for the selected server, you are prompted to log in. △

3 Select the name of the SAS library that points to the tables that you are importing from the current SAS Workspace Server. Click **New** to create a new library. Click **Next** to see a list of SAS tables in the selected library.

4 Select the table(s) that you want to load into the metadata, and then click **Next**.

5 Click **Finish**.

Note: These panels might not be available if you have already defined a connection to the server on the Options panel. △

Note: Subsequently, you can delete the defined table(s) . In SAS OLAP Cube Studio, complete the following steps:

1 Open SAS OLAP Cube Studio.

2 In the **Navigation Tree**, expand the **Repositories** folder.

3 Expand the **default repository** folder.

4 Expand the **Physical Table** folder to see a list of OLAP tables that are defined in the default repository.

5 Right-click the table that you want to delete. Click the **Delete** option.

△

Registering Tables without a SAS Workspace Server

To register SAS data sets in the SAS Metadata Repository without a SAS Workspace Server, you can use the meta libname engine's METAOUT=META option. To do so, complete the following steps:

1 Register a library in the metadata repository by using SAS Management Console. In this example, a library definition called *mlelib* is added to the repository *proc*. It is defined as a base engine library with a path of **c:\olaptestfiles**.

2 Assign the libraries in your SAS session.

You can assign a meta library that points to your new library definition:

```
libname mymlelib meta repname=proc library=mlelib
ipaddr="d5014.us.sas.com" port=9999
! protocol=BRIDGE;
```

The libref *mymlelib* is successfully assigned as follows:

```
Engine:        META
Physical Name:
```

You can assign a base library whose path matches the metadata path.

```
libname mybaslib "c:\olaptestfiles";
```

The libref *mybaslib* is successfully assigned as follows:

```
Engine:        V9
Physical Name: c:\olaptestfiles
```

If you are using user-defined formats, set them up now.

```
options fmtsearch = (mybaslib);
```

Note: The libraries do not have to be base engine libraries. You can define an Oracle library in the metadata and assign a library by using the Oracle library in your SAS session. It is important that both libraries resolve to the same LIBNAME statement. △

3 Use METAOUT=META to register the data set in the metadata. The data set option METAOUT=META tells the MLE engine that you want to create only metadata and not a physical file, as it already exisits.

```
data mymlelib.cars(metaout=meta);
set mybaslib.cars(obs=0);
run;
```

Note: There were zero observations read from the data set mybaslib.cars. The data set mymlelib.cars has zero observations and nine variables. △

Note: DATA statement used (total process time):

real time – 1.06 seconds

cpu time – 0.03 seconds

△

The MLE engine has just written all the metadata from the data set to the metadata repository. Note the use of the (obs=0) data set option for the input data set. If you leave the option off, the metadata will be registered anyway, but each observation of the data set will be read unnecessarily.

4 Use PROC OLAP to build the cube by using the new registration. Use the new metadata registration to build your cube:

```
proc olap data=mymlelib.cars cube=mddbcars path="c:\v9olap.proc";
METASVR  repository=proc olap_schema="proc";
dimension date hierarchies=(date) sort_order=ASCENDING;
hierarchy date levels=(dte);
dimension cars hierarchies=(cars) sort_order=ASCENDING;
hierarchy cars levels=(car color);
dimension dealers hierarchies=(dealers) sort_order=ASCENDING;
hierarchy dealers levels=(dealer dest);
measure sales_sum column=SALES stat=sum format=dollar15.2;
measure sales_n column=SALES stat=n format=12.0;
run;
```

Note: The number of NWAY records is 20. The cube mddbcars was created successfully. △

Note: PROCEDURE OLAP used (total process time):

real time – 1.06 seconds

```
cpu time – 0.12 seconds
```

△

Setting Up System Access Permissions for SAS OLAP Server

As one step in configuring the SAS Metadata Server, you set up system access permissions. It is also necessary to set up system access permissions for the SAS OLAP Server. Depending on your Windows operating system, select the **User Rights** or **User Rights Assignment** option from the Windows administrative tools folder. Select and add these policy settings for access permissions:

- Act as part of the operating system (Windows NT and Windows 2000 only)
- Log on as a batch job. Assign the *Everyone* identity as server accessor. This enables all users and applications to access the OLAP server.

  Note: As an alternative, you might consider defining a SAS Users group and assign the Log on as batch job user right to this group instead. △

Adding an OLAP Server to a SAS Metadata Repository

The following steps are necessary for querying cube data. However, they are not required to create cubes in SAS OLAP Cube Studio. These steps enable you to add an OLAP server to a SAS Metadata Repository:

1. Verify that the correct resource templates are loaded.
2. In SAS Management Console, select **Server Manager**. Right-click, and then select **New Server**.
3. In the New Server Wizard window, select **SAS Application Server**, and then define the new server. Enter the name, description, and server properties.
4. Select the OLAP Server type that you are creating, and then define the default configuration settings.

   Note: The default configuration generates a default OLAP schema and assigns that schema to the server that is being defined. However, you can create and define additional schemas with the property sheet for the SAS application server. You can also use SAS OLAP Cube Studio to create and assign a new schema. △
5. On the final screen, click **Finish** to create the server.

You can define any needed additional OLAP schemas in a SAS Metadata Repository. In addition to an OLAP server definition, you can define an OLAP schema in the active metadata repository. The OLAP schema specifies which group of cubes that an OLAP server can access. Each OLAP schema can be accessed by multiple OLAP servers. However, each OLAP server has access to only one OLAP schema.

To create an OLAP schema in SAS OLAP Cube Studio, complete these steps:

1. Launch SAS OLAP Cube Studio, and connect to the SAS Metadata Server.
2. Select

   File ▶ New OLAP Schema

   to launch the OLAP Schema wizard.
3. In the General window, enter the schema name and description.
4. (Optional) In the Server Assignment window, specify the OLAP servers that can access the schema. Click **Next**.

   Note: If you choose not to specify the servers by using the OLAP Schema wizard, you can add that information later by modifying the schema's property sheet. △

5 In the Finish window, click **Finish** to create the schema.

For detailed help on completing the OLAP Schema wizard, click the **Help** button in any wizard window.

Note: You can also create OLAP schemas by clicking **New** in the General window of the Cube Designer wizard. △

Creating and Modifying the SAS OLAP Server Script

You can create a script for the OLAP server startup with the SAS OLAP Server Monitor plug-in for SAS Management Console. Here you enter the server identification and user information that is necessary to start an OLAP server.

Note: The SAS OLAP Server Monitor plug-in for SAS Management Console must be installed in addition to SAS Management Console and SAS OLAP Cube Studio. △

In the SAS Management Console navigation tree, scroll to the Monitoring plug-in folder, and then select the existing OLAP server that you want to create a server script for. Right-click on the selected OLAP server, and then select the **Generate server script** option. This loads the Command window. In the various windows of this wizard you enter the following information:

Command Window

Command	specifies the SAS command that is used to launch the SAS OLAP Server. The Command field is required. The command text is enclosed in double quotation marks when it is written to the script file.
SAS system options	specifies SAS system options. For example, you might enter **-nodate -linesize 80**, which sets the NODATE and LINESIZE system options. This field is optional.
Object server parameters	specifies any additional server startup parameters. It can contain any object server parameters that are documented in the SAS server documentation. For example, you might enter **multiuser**. The Object server parameters field is optional.

Metadata Window

Host	this read-only field identifies the machine that hosts the active metadata server.
Port	this read-only field identifies the port for the active metadata server.
Repository	this read-only field identifies the name of the repository that you use in the active metadata server. The repository must contain the definition of the SAS OLAP Server that you want to start.
Include the current user ID and password	select this check box to include in the script the user ID and password that were used to access the active metadata server. By selecting this check box, you are adding the METAUSER and METAPASS options to the startup script.
Encrypt the password if included?	is selected by default. If selected, the password is encrypted before it is written to the file.

Files Window

AUTOEXEC file name — specifies the location and filename of the AUTOEXEC file, which contains SAS statements that you want to process when the SAS OLAP Server is initialized. You can enter a fully qualified path or a filename that is relative to the directory where the server is started. For example, you might enter **c:\v9olapserver\autoexec.log**. Users commonly use an AUTOEXEC file to include LIBNAME statements that are required for cubes that contain data from summarized sources. This field is optional. If included, the filename is enclosed in double quotation marks when it is written to the script file.

Log file name — specifies the name of a log file for the -altlog option. You can enter a fully qualified path or a filename that is relative to the directory where the server is started. For example, you might enter **c:\v9olapserver\serverlog.log**. Server messages are written to this file. This field is optional. If included, the file name is enclosed in double quotation marks when it is written to the script file.

Paths Window

SAS user root — specifies the name of the SAS user root (SASUSER) file. You can enter a fully qualified path or a filename that is relative to the directory where the server is started. For example, you might enter **c:\v9olapserver\sasuser**. This field is optional. If included, the path name is enclosed in double quotation marks when it is written to the script file.

SAS work root — specifies the name of the SAS work root (WORK) file. You can enter a fully qualified path or a filename that is relative to the directory where the server is started. For example, you might enter **c:\v9olapserver\saswork**. This field is optional. If included, the path name is enclosed in double quotation marks when it is written to the script file.

File Window

In this window, you enter the name of the server script file. In the File name field, enter a fully qualified name for the file. The filename field is required.

Finish Window

This window displays the server script command string as defined by the entries that you made on the other wizard panels. The command is written to the file that you specified in the File window. Click **Finish** to save the file.

Here is an example of a server script:

```
sas -nosplash -noterminal -objectserver -objectserverparms
"server='omsobj:LogicalServer/A32JZRHX.AV0000B5 "
-METASERVER finance.us.abc.com -METAPORT 9999
-METAPROTOCOL bridge -METAREPOSITORY "RepositoryOne"
-METAUSER fburns -METAPASS xxxxxxxx
-autoexec " c:\v9olapserver\autoexec.log"
-altlog " c:\v9olapserver\serverlog.log"
-sasuser "c:\v9olapserver\sasuser"
-work "c:\v9olapserver\saswork"
```

If you do not have the SAS OLAP Server Monitor plug-in installed, you can manually create a SAS OLAP Server batch job by completing these steps:

1 In SAS Management Console, under Server Manager, select the OLAP server, and then select

OLAP Server ► Logical OLAP Server

Right-click, and then select **Properties**.

2 On the General tab, highlight the ID number and copy it to the Windows clipboard by pressing Ctrl-C.

3 In the OLAP server batch file, replace the listing for the server with the ID number that you highlighted and copied.

4 Replace METAUSER and METAPASS with the required user ID and password.

Starting the SAS OLAP Server as a Service

You can set up the OLAP server to start as a service by using the SAS Service Configuration Utility (SSCU). Follow these steps:

1 Use an ASCII text editor or the SSCU and create a file named SASOMA.INI in the server directory that contains the following information:

```
[SASOLAP]
ServiceName="SASOLAP"
DisplayName="SAS OLAP Server"
Dependencies=""
BinaryPathName="["c:\program files\sas\sas 9.1\sas.exe"
-nosplash -noterminal -altlog
"olapserver.log" -objectserver -objectserverparms "
server=omsobj:LogicalServer\A58O0U1V.AT000002 " -METASERVER localhost
-METAPORT 9999 -METAPROTOCOL bridge -METAREPOSITORY "foundation"
-METAUSER userid  -METAPASS password ]"
StartType=SERVICE_AUTO_START
ErrorControl=SERVICE_ERROR_NORMAL
Interactive=FALSE
AccessChgCfg=TRUE
AccessInterrogate=TRUE
AccessPauseCont=FALSE
AccessQryCfg=TRUE
AccessQryStatus=TRUE
AccessStart=TRUE
AccessStop=TRUE
AccessUserDefCtrl=TRUE
WorkDir="c:\olapserver"
ServiceStartName=userid
Password=<password>
```

This example uses the —META* options to identify the SAS Metadata Server that the SAS OLAP Server connects to. See “SAS Metadata System Options” in *SAS Open Metadata Architecture Reference* for information about other ways to identify the SAS Metadata Server.

If your SAS Metadata Server is running as a service, you can define a dependency between the SAS OLAP Server and the SAS Metadata Server by entering the SAS Metadata Server ServiceName in the Dependencies field.

Note: The BinaryPathField contains the command that is used to launch the server. Be sure to specify the correct LogicalServer ID. Or you can use the Generate Startup Script feature of the SAS OLAP Server Monitor to create this command. △

2 Issue the following command to uninstall any existing .INI files:

```
"C:\Program Files\SAS\9.1\core\sscu\SASServiceMngr.exe" /remove SASOLAP
```

3 Restart your machine.

4 Issue the following command to install the SASOLAP.INI file:

```
"C:\Program Files\SAS\9.1\core\sscu\SASServiceMngr.exe" C:\sasolap.ini
```

5 Issue a NET START command or restart your machine.

For more information, see "Starting SAS as a Windows Service" in the *SAS Companion for Microsoft Windows*.

Defining Encryption for SAS OLAP Server

Encryption is a security measure that transforms plaintext into ciphertext. The ciphertext is translated back to plaintext when the appropriate decryption key is applied. Encryption can be used by all IOM-based servers. SAS OLAP Server communicates with a SAS Metadata Server. If the metadata server is started with encryption turned on, SAS OLAP Server must be able to encrypt as well. Two encryption services that are supported by SAS are *SAS Proprietary* and *SAS/Secure*.

SAS Proprietary	is a fixed encoding algorithm that is included with Base SAS software and is supported under the OpenVMS Alpha, z/OS, UNIX, and Windows operating environments. It requires no additional SAS product licenses. The SAS proprietary algorithm is strong enough to protect your data from casual viewing.
SAS/Secure	is an add-on product that provides encryption algorithms in addition to the SAS proprietary algorithm. SAS/SECURE software requires a license, and it must be installed on the metadata server host and each client host that will use the encryption algorithms. SAS/Secure is required if you want to use an encryption algorithm that is different from SAS Proprietary. Some industry-standard algorithms that require a SAS/Secure license include RC2, RC4, DES, and TripleDES.

Here are the SAS options that set encryption services attributes:

NETENCRYPT=YES|NO
or
NETENCRYPT | NONETENCRYPT
Set this option for both the local and remote hosts. For the remote host, this option specifies that encryption is required for each connection from a local host SAS session. On the local host, this option specifies that the local host must connect only to a remote host that supports encryption.

By default, encryption is used if the NETENCRALG= option is set and if both the local and remote hosts are capable of encryption. If encryption algorithms are specified, but either the local or the remote host is incapable of encryption, then encryption is not performed.

The NETENCRYPTALGORITHM option specifies that encryption is required by any client that accesses this server. The metadata server and clients must specify at least one common algorithm in the NETENCRYPTALGORITHM option, or the server connections will fail.

NETENCRALGORITHM= ("algorithm1", "algorithm2", ...)
the local and remote hosts must have an encryption algorithm in common. If you specify the option in the remote host session only, then the local host attempts to select an algorithm that was specified on the remote host. If you also set the

option on the local host and specify an algorithm that is not specified on the remote host, then the attempt by the local host to connect to that remote host fails. Valid values for this option are RC2, RC4, DES, TripleDES, and SAS Proprietary.

CLIENTENCRYPTIONLEVEL=NONE|CREDENTIALS|EVERYTHING
This is an OBJECTSERVERPARAM option that is set in conjunction with the NETENCRYPTALGORITHM= option. It specifies the degree of encryption to use when making outbound calls.

NONE	Nothing is encrypted. Clients transmit their user IDs and passwords as plaintext.
CREDENTIALS	Clients encrypt their user IDs and passwords.
EVERYTHING	encrypts all client-server communications.

This option is used only by the bridge protocol engine.

You should store information about the encryption options in your metadata server metadata definition and in your OLAP server metadata definitions. The settings that are specified in the metadata should match those that are specified in the start commands. You must change the scripts for the metadata server and OLAP server to include the objectserverparam CLIENTENCRYPTIONLEVEL and the SAS System Option -NETENCRYPTALGORITHM set to the appropriate values.

To start an OLAP server by using encryption, you must first define the OLAP server in SAS Management Console:

1. In the New Server Wizard - SAS Server Configuration window, select **Custom Configuration**.
2. In the New Server Wizard - Connection Options window, select **Advanced Options**.
3. In the Advanced Options window, select the **Encryption** tab.
4. Select the **Server Encryption Algorithms** that you want the server to recognize. Select the Required Encryption Level option, **Everything**.
5. Select **OK** when finished. Finish creating the new OLAP server in the New Server Wizard.

This establishes the encryption settings for the OLAP server that you are creating.

To start the server in a batch file, you need to include the NETENCRYPTALGORITHM system option and the CLIENTENCRYPTIONLEVEL OBJECTSERVERPARMS option. The startup command would look like this:

```
c:\program files\sas\sas 9.1\sas.exe"  -nosplash -noterminal -netencryptalgorightm
SASProprietary -objectserver -objectserverparms
"server=omsobj:LogicalServer\A50H71JY.AQ00002T  iomlevel=1
clientencryptionlevel=everything" -METASERVER d3635.na.sas.com -METAPORT 9999
-METAPROTOCOL bridge  -METAREPOSITORY "foundation" -METAUSER "carynt\sasalw"
-METAPASS "{sas001}RGVhZDA1MTUwMg==" -log "olapserver.log" -sasuser "sasuser
```

Note:

- See “Encrypting Client-Server Communication” in the *SAS Metadata Server: Setup Guide* at **http://support.sas.com.**
- See “Specifying Server Encryption Settings for IOM Bridge Connections” in the *SAS Integration Technologies Administrator's Guide* at **http://support.sas.com**.
- For a description of the RC2, RC4, DES, and TripleDES algorithms, see “Data Security” in the *SAS/SHARE User's Guide.*

△

Cleaning Up Temporary Performance Data Files

When the OLAP server is terminated correctly, various files are automatically deleted. Occasionally, the OLAP server might terminate incorrectly. When this happens, some performance data-related files might be left on the host rather than being cleaned up.

There are two types of temporary performance files:

Temporary performance data files that are used to build the aggregates from SAS by using PROC OLAP

These include the temporary files

- that are used to create the *aggregates*. These files are created in the cube subdirectory (within the OLAP PATH= defined directory) or the optionally defined DATAPATH= aggregate directory.
- that are used to create the *aggregate indexes*. These files are created in the working directory for the performance engine.

Temporary performance data files that are used by the OLAP server to query the aggregates.

The OLAP server uses temporary performance data files to complete queries to the aggregates. The temporary files are created in the working directory for the performance engine.

Java Virtual Machine and SAS OLAP Server

As a general rule, SAS OLAP Server always tries to load the *Java Virtual Machine (JVM)* when it starts. This is in order to provide support for MDX external functions. If SAS OLAP Server is unable to load the JVM, it sends the following message and continues to start up: **`WARNING: The Java Virtual Machine is not loaded. External functions are not available.`** At this point, the SAS functions are working, but the MDX external functions are disabled.

If you are not planning to use external functions, then you can safely ignore the warning. Some common reasons why you might not be able to load the JVM include the following:

- There is no JVM installed on the platform that SAS OLAP Server is installed on.
- An unsupported version of the JVM is installed on the platform that SAS OLAP Server is installed on.
- The JVM is incorrectly configured (some JVMs require that certain environment variables be set up in order to load).

Note: For further information about external functions and the JVM, see "External Functions" in "MDX Queries and Syntax" in the *SAS OLAP Server MDX Guide*. For information about installing the Java Runtime Environment, see the *SAS Third-Party Software Components Volume 1* CD. △

Monitoring OLAP Server Performance

SAS OLAP Server performance is monitored and logged with the Application Response Measurement (ARM) interface. Traditionally, ARM enables system

administrators to monitor application executions, run times, performance, and completion. SAS OLAP Server uses ARM to monitor

- application behavior
- user behavior and usage
- server loads
- cube optimization (query response time)
- cube metrics—counts of connections and queries.

For more information, see "Monitoring Performance Using Application Response Measurement (ARM)" in *SAS Language Reference: Concepts*.

Changing an OLAP Server Configuration

To change the server configuration, complete the following steps:

1. Within SAS Management Console, expand the nodes under the **Server Manager** until you can see your OLAP server component. The tree structure lists the SAS application servers first, then the logical servers that are defined for the application server, and finally the server components (OLAP and Metadata Profile).

   Note: The logical server definitions enables you to pool servers so they can be treated as a single server. OLAP servers do not support pooled servers in SAS 9.1. The server components represent the actual machines that the OLAP server is running on. △

2. After you find the desired OLAP server component in the tree, right-click and display the properties for the server by selecting **Properties** from the drop-down menu. This opens the Properties window, which contains panels for
   - General (OLAP server name, ID, and description)
   - Options (and Advanced Options button)
   - Notes
   - Extended Attributes
   - Authorization (read, write, create permissions)
3. When you select **OK**, the information that you entered is written to the server metadata.

Configuring Server Options

After you start and initialize the OLAP server, you can configure various server options such as turning its cache on and defining how many cubes can be in a cache. In the

Server Manager ► OLAP Server Properties ► Options

panel, you can select the **Advanced Options** button, which enables you to update various server options. This window contains panels for

- Cache
- Server
- Debug Query

- Debug Server
- Journal.

The following table outlines all of the OLAP server options that you can configure.

Table 2.1 Server Options

Server Option	Type	Description
ARM FLAG	integer	ARM values set (desired values from Application Response Measurement (ARM) Options on page 31)
ARM LOCATION	string	Filename of ARM log
CACHE ACTIVE	integer	Cache on/off switch 1=on, 0=off
DEBUG FLAG	integer	Debug values set
GROUPBY ROW CACHE	integer	Amount of memory available for parallel group-by operations. The default and minimum is 256 megabytes
MAX CUBE CACHE	integer	Maximum number of cubes (and their metadata) that can be cached by the server
MAX FLAT MEMORY	integer	Maximum amount of memory allowed for Flattened Data Sets processing
MAX FLAT ROWS	integer	Maximum number of flattened rows allowed for Flattened Data Sets
MAX SEGRATIO	integer	Specifies as a percentage, the upper limit for the number of index segments in which a value can be located in order to perform pre-evaluation on the WHERE expression processing. The default is 75, which indicates that the value should be in 75% of the index segments or less. The range of valid values is 0-100. If set to 0, pre-evaluation never occurs; if set to 100, pre-evaluation always occurs
MAX SET SIZE	integer	Maximum number of members in a set
MAX THREADS	integer	Maximum number of region execution threads
OLAP SCHEMA	string	Name of OLAP server in SAS Metadata Repository
SAS METADATA SERVER HOST	string	SAS Metadata Server IP address

Server Option	Type	Description
SAS METADATA SERVER PASSWORD	string	SAS Metadata Server authentication password
SAS METADATA SERVER PORT	string	SAS Metadata Server port number
SAS METADATA SERVER PROTOCOL	string	SAS Metadata Server protocol
SAS METADATA SERVER REPOS	string	SAS Metadata Server Repository
SAS METADATA SERVER USERID	string	SAS Metadata Server authentication user ID
SIZE OF DATA CACHE	integer	Megabytes to use for data caching
SPDE MAX THREADS	integer	Number of threads SPD should spawn. The range of valid values is 1-8
WORKPATH	string	Path for SPDE temporary work files. It defaults to the SAS working directory, or, if that is not given, to the current cube directory

The following table outlines all of the OLAP server debug options that you can configure.

Table 2.2 Debug Options

Debug Option	Value	Description
DBG_CACHEEXEC	0x00400000	Prints the cache-executing flow
DBG_CACHELOAD	0x00200000	Prints the cache-loading flow
DBG_CACHEMEM	0x00080000	Prints the cache memory changes
DBG_CACHEPICK	0x00100000	Prints the cache chooser or picker flow
DBG_CACHERANGE	0x00800000	Prints the cache iterator ranges
DBG_EXEC	0x00000080	Prints the query execution flow
DBG_FLATRS	0x00000040	Prints the flattened row set
DBG_MDX	0x00000001	Prints the input MDX statement
DBG_MDXSEM	0x00010000	Prints the MDX semantic action processing flow
DBG_MDXTOK	0x00000800	Prints the assignment of a token MDX
DBG_OMR	0x00001000	Prints the input/output of OMR requests
DBG_OVM	0x00000400	Prints the generated OVM code streams
DBG_OVMEXEC	0x00008000	Prints the OVM execution flow
DBG_PARSE	0x00000002	Prints the MDX parse tree
DBG_RANGES	0x00000008	Prints the query ranges
DBG_REGIONS	0x00000020	Prints the query regions

Debug Option	Value	Description
DBG_SECURITY	0x00040000	Prints the security information
DBG_SETS	0x00000004	Prints the contents of generated sets
DBG_SPDE	0x00000100	Prints SPDE query information
DBG_SPDEROWS	0x00000200	Prints SPDE queried rows
DBG_SQL	0x00020000	Prints SQL statements generated by the classic SAS plug-in
DBG_WIREFRAME	0x00000010	Prints the query wireframes
DEL_DBGJNL	0x00002000	Deletes the session debug journal when the session is terminated
DEL_SVRJNL	0x00004000	Deletes the server journal when the server is terminated
DBG_AVGOFAVGS	0x01000000	Computes the average of averages by adding all the average values and dividing by the number of averages.

The following table outlines all of the OLAP server ARM options that you can configure.

Table 2.3 Application Response Measurement (ARM) Options

ARM Option	Value	Description
DATA_QUERY	0x00000080	Records each individual data retrieval from stored cube aggregations or from the cache
MDX_QUERY	0x00000008	Records cube name and result set size (in cells) for each query
MDX_STRING	0x00000800	Records the actual MDX query string
OLAP_SESSION	0x00000001	Records which user was logged in to which server for how long

Optimizing OLAP Server

SAS Management Console affects server query performance in three ways, by controlling the amount of

- cube metadata that is cached
- cube aggregations that are cached
- parallelism that can be used to execute a query.

Cube Cache

The cube cache helps speed up queries by saving an in-memory copy of a cube's metadata. A cube's metadata contains information that is necessary to parse and plan an MDX query for the cube. The metadata does not include any disk-resident

aggregates. As the server processes each query, the server first checks the cube cache to determine if the cube metadata is in memory. If it is, the server uses the cached metadata. If it is not, it needs to load the metadata.

SAS Management Console can control the number of cubes that can be cached by the server. Increasing the number of cubes increases performance, but it uses more memory. The cube cache is an LRU (least recently used) cache. As the cache becomes full, cubes are removed based on the usage.

Note: The default number of cubes cached is 20. △

Data Cache

The data cache helps speed up queries by saving copies of the underlying aggregates in memory from previous queries. As the server processes each query, it first checks the in-memory aggregates to see if the query contents are there. If the query contents are there, the query processor uses those aggregates instead of accessing the disk. SAS OLAP Cube Studio provides a plug-in to SAS Management Console that enables you to define and start a server, assign a schema, and configure your data cache. See "Installing and Configuring SAS OLAP Server" on page 8 for further information about OLAP server configuration and setup. For more information about SAS Management Console, see the *SAS Management Console Administrator's Guide*.

Enabling the Data Cache

The OLAP Server data cache is enabled by default. However, if the data cache has been previously disabled, then it can be enabled in SAS Management Console. To enable data cache in SAS Management Console, complete these steps:

1. Navigate to **Server Manager** in the navigation tree.
2. Select **OLAP Server** (not the logical OLAP Server), and then right-click. Select **Properties**.
3. In the OLAP Server Properties window, select the **Options** tab, and then select the **Advanced Options** button.
4. Select the **Cache** tab.
5. Click the **Cache Active** check box to enable the data cache. Select **OK**.

Disabling the Data Cache

To disable the data cache through SAS Management Console, complete these steps:

1. Navigate to **Server Manager** in the navigation tree.
2. Right-click **OLAP server**. Select **Properties**.
3. In the OLAP Server Properties window, select the **Options** tab, and then the **Advanced Options** button.
4. Select the **Cache** tab.
5. Click the **Cache Active** check box to disable the data cache. Select **OK**.

Determining Memory Size for the Data Cache

When you establish data cache memory, you should use no more than 10 percent of the system's virtual memory. To get the recommended amount of memory for caching aggregates, follow this formula:

```
(Number of possible cells) * (Number of stored statistics) * (8 bytes) * 4
= number of megabytes of memory to allocate for the OLAP data cache
```

To calculate the possible number of cells, multiply the number of values in each level of each dimension or hierarchy that is stored in the aggregate. For example, if you use the PRDMDDB cube, then the NWAY aggregate has three dimensions (Time, Geography, and ProductLine).

- In the Time dimension, the NWAY aggregate has three levels (Years, Quarters, and Months). Multiply the number of Years (2) * Quarters (4) * Months (12) to get 96 possible cells for Time.
- In the Geography dimension, the NWAY aggregate has three levels (Country, Region, and Division). Multiply Countries (3) * Region (2) * Division (2) to get 12 possible cells for Geography.
- In the ProductLine dimension, the NWAY aggregate has two levels (ProductType and Product). Multiply ProductType (2) * Product (5) to get 10 possible cells for ProductLine.

The total number of possible cells is Time (96) * Geography (12) * ProductLine (10). This results in a final count of 11,520 possible cells.

To calculate the number of stored statistics, add the stored statistics for each measure. For example, if you have two measures named Actual and Budget,

- the measure Actual has six stored statistics: Count, Sum, Max, Min, USS, and X.
- the measure Budget has two stored statistics: Count and Sum.

These stored statistics measure results in eight total stored statistics (six and two).

Multiplying the above values for possible cell count by the number of stored statistics results in a total number of possible cells (11,520) * the total stored statistics (8) * 8 * 4 = 2,949,120 bytes for the NWAY aggregate. The recommended cache size is 3 * 4 = 12 MB of memory to allocate for the OLAP data cache.

Note: The default MAXMEM size is 16 megabytes (MB). △

Note: As a general rule, you can expect that more than one cube will be active in a given server invocation, so you need to have space for multiple aggregates across multiple cubes. In addition, although the NWAY is a single aggregate of this size, you need space to handle multiple aggregates in the same cube.

A basic rule is to multiply the NWAY size by 4 in order to reserve enough memory to hold the NWAY and its derivative aggregates. △

Note: The data cache size can be set in SAS Management Console. At the Server Manager, complete these steps:

1. Select **OLAP server** (not the logical OLAP Server), and then right-click. Select **Properties**.
2. In the OLAP Server Properties window, select the **Options** tab, and then the **Advanced Options** button.
3. Select the **Cache** tab.
4. Set the data cache to the needed size. Select **OK**.

△

Number of Execution Threads

A single MDX query is planned and executed into several MDX subqueries. Executing the subqueries in parallel allows for improved query performance. SAS Management Console controls the maximum number of threads that can concurrently execute a single MDX query. To set the number of execution threads through SAS Management Console, complete these steps:

1 Navigate to **Server Manager** in the navigation tree.
2 Select **OLAP server**, and then right-click. Select **Properties**.
3 In the OLAP Server Properties window, select the **Options** tab, and then the **Advanced Options** button.
4 In the Advanced Options window, select the **Server** tab.
5 Enter the number of threads in the **Maximum number of region execution threads** field. Select **OK**.

Increasing this value can increase parallelism for those queries that use the full number of threads that are available. However, limited system resources must be shared by all MDX queries. As a result, on a system with many users, the maximum number of execution threads should be judiciously set to allow queries to perform without having a single query monopolize the system resources.

Typically, a reasonable range for the number of execution threads per MDX query would be

- a minimum of two execution threads.
- a maximum number of execution threads, which is determined by the number of processors on the system multiplied by two. This is the default value.

Monitoring and Administering Sessions—SAS OLAP Server Monitor Plug-In

The SAS OLAP Server Monitor plug-in is an additional component of SAS Management Console. It enables you to connect to an OLAP server, generate a server startup script, monitor sessions and queries, and stop a server.

To access the SAS OLAP Server Monitor, expand **Monitoring** in the navigation tree until you can see the SAS OLAP Server Monitor and the Logical OLAP Server. Select the **Logical OLAP Server**. From here you perform the following administrative functions:

1 Connect to the OLAP server and subsequently
   - view sessions and queries for an active OLAP server
   - close individual active sessions.
2 Disconnect from the OLAP server.
3 Stop the OLAP server from processing queries.
4 Generate a server script for the OLAP server.
5 View Properties for the Logical OLAP Server.

These functions are available from the Actions menu or the Toolbar.

When you connect to a server, you see a list of sessions that are running queries against cubes that are stored on the OLAP server machine that is associated with the selected logical server.

View Sessions and Queries — The active sessions for the server are listed in a table on the right side of the screen. Click the plus sign (+) next to the logical server name to see a list of active sessions. Sessions are identified by the ID of the user who initiated the session. If the server is not secured, then the owner displays as <unknown>. The following information is displayed about sessions:

- session name

- description
- session owner
- time (in seconds) that the server has been inactive
- number of open results sets.

If you select a server and it already has an active connection, then the active sessions will be listed in the table. If there are no active sessions (either no connections exist, or there is no active connection), then an error message displays in the table.

To list the active queries for a session, select the session in the navigation tree in the display area of SAS Management Console. For each active query, SAS Management Console displays the

- type of query (unspecified, multidimensional, or flattened)
- MDX string.

Close Sessions — To close a session, select the session in the table and select **`Close session`** from the drop-down menu.

You can create a server startup script with the Generate Server Script Wizard that is available from the

Monitoring ▶ SAS OLAP Server Monitor plug-in

With this script you define information that is needed to start a server including the following:

- command to launch the OLAP server
- object server parameters
- host, port, and repository information
- user ID and password (optional)
- autoexec and log filenames
- name of SAS user and work root files
- server startup filename.

See "Creating and Modifying the SAS OLAP Server Script" on page 22 for further information.

Note: If you are running a secured OLAP server, then users who connect to view sessions or stop the server must have administrative privileges defined for their login in the metadata. △

Securing Cubes

When you administer the OLAP server, you must assign and maintain security permissions for users. The User Manager plug-in for SAS Management Console establishes users, groups, and login access. The Authorization Manager plug-in for SAS Management Console is used to grant read and readmetadata permissions for cubes. These restrictions are set for individual users, multiple users within a group, and groups.

Assigning Users and Groups in the User Manager Plug-In

In SAS Management Console, you maintain user, group, and login information in a metadata repository. The User Manager plug-in enables you to register a unique

metadata identity for each individual and group by creating user definitions and group definitions. For SAS Management Console purposes, a user is someone who is registered in a SAS Open Metadata environment. Until a particular user is represented in the metadata environment by an identity object that owns at least one login, you cannot select that user when you configure access control or change management. After a user is established, the user can be assigned to a group. A group can also contain other groups. This allows for more streamlined control of user privileges. An administrator can also establish multiple logins for each user or group. This enables each user or group to connect to all of the resources that are registered in the metadata environment by using a single identity.

To assign users or groups in SAS Management Console, select **User Manager** from the navigation tree. Right-click and select **New**, and then **User** or **Group**. This opens the Properties dialog box for either the new user or new group. You can assign new users, groups, and user IDs and passwords. See the User Manager Help in SAS Management Console for further information.

Authorization Manager Plug-In

The Authorization Manager enables you to set permissions for objects in the SAS Metadata Repository. If you are using a secured metadata server, you must have read permission granted in order to read SAS OLAP cubes. *Read* permission is required to view cube data. The read permission is enforced for OLAP queries by the OLAP Server code. The *administer* permission is enforced by the IOM code for the stop action or list session action of the OLAP Server Monitor.

Readmetadata and *Writemetadata* permissions are enforced by the metadata server. The *Readmetadata* permission is required to see the cubes that are listed in the navigation tree. OLAP then inherits the permission enforcement.

Note: For further information about SAS Metadata Server permissions, see the *SAS Metadata Server: Setup Guide* at **http://support.sas.com.** △

In the SAS Management Console navigation tree, select

Environment Manager ► Authorization Manager ► Resource Management ► By Location

The By Location folder lists cube resources according to their server locations. You can choose cubes, schemas, dimensions, levels, hierarchies, and measures that exist for an OLAP cube. You can then set permissions for the selected OLAP cubes, schemas, and other resources.

Select a cube or other resource and complete the following steps:

1. Right-click and select **Properties**. This opens the Properties dialog box.
2. In the Properties dialog box, select the **Authorization** tab.
3. Use the **Add** button to assign users to the **Names** list box.
4. In the **Permissions** list, select the **Read** and **ReadMetadata** check boxes in the Grant column. Do this for each user that you want to assign read permissions to.

   Note: The SAS OLAP Server supports only *read* permission for cube data. △

See the Authorization Manager help in SAS Management Console for further information.

Access Control Templates

An access contol template (ACT) consists of a list of users and groups that are represented in the metadata environment. An ACT indicates, for each user or group, whether permissions are granted or denied. The permissions information in an ACT is reusable and can be applied to multiple resources.

A default ACT is automatically created when you register a repository. During the permission evaluation process, the default ACT is examined to determine whether any access controls that pertain to the requesting user and the requested action have been specified. The **`Default ACT`** is located in the SAS Management Console navigation tree under

Authorization Manager ► Access Control Templates

If a default ACT is not listed, then you can select the ACT that you want to designate as the default ACT for the current repository. To make an ACT the default, from the main menu select

Actions ► Repository ACT

Note: The default ACT grants readmetadata and writemetadata permission to the PUBLIC implicit group. It is important to note that if you run a secured metadata server but do not add users or groups, you should grant the read permission in the default ACT template. △

Permission Condition for Dimensions

OLAP cube dimensions have specific permission conditions that are set. A permission condition is used to specify member-level security for a dimension in a cube. When you specify a permission condition for a dimension, the selected user or group has read access to only those members for which the condition evaluates as true. A permission condition is defined by specifying an MDX expression that results in the return of a member or a set of members. In addition, valid MDX expressions are limited to those that would return a member or set of members from the descendants of the dimension in question (not a union of members from other dimensions).

For example, you are building a cube that has the following structure and parameters:

- The cube contains the dimensions Dealers, Cars, Date, and Measures.
- The Dealers dimension contains the levels All, Dealer, and Destination.
- The permission condition is applied to the Dealers dimension.
- The Dealer level contains the members: Finch, Jones, and Smith.

Here are sample MDX expressions that would return members from the various levels of the Dealers dimension:

- ```
  {[Dealers].[Dealer].members}
  ```

  This expression enables the user to see only the members of the Dealer level (Finch, Jones, Smith). Using this expression would *prevent* the user from seeing any of the members of the Destination level.

- ```
  EXCEPT({[Dealers].members},  [Dealers].[Dealer].members)
  ```

  This expression enables the user to see all members from the Dealers dimension *except* the members of the Dealer level (in other words, the members of the Destination level).

- ```
  EXCEPT({[Dealers].members},  [Dealers].[All Dealers].Finch)
  ```

  This expression enables the user to see all members from the Dealers dimension *except* the member Finch from the Dealer level and its children (from the Destination level).

- ```
  {[Cars].[Car].members}
  ```

  This expression, placed on the Dealers dimension, is *invalid*. This results in the user seeing *no* members of the dimension. The same expression, placed on the Cars dimension is valid and would enable the user to see only the members of the Car level.

In SAS 9.1, you can set permission conditions only for dimension objects. Additionally, these conditions limit only read access. See “Set Permission Conditions” in the Authorization Manager help in SAS Management Console for further information.

Invoking a Secured Metadata Server

Currently, SAS Management Console provides a default option on the server startup command line that indicates whether the server should be secured or not. If a non-secured metadata server is desired (this is not recommended), add the *nosecurity* parameter to the list of **`Object server parameters`**. This is located in the Generate server script/Command window in the OLAP Server Monitor plug-in.

```
-objectserverparms "nosecurity protocol=bridge port=9999
classfactory=2887E7D7-4780-11D4-879F-00C04F38F0DB"
```

Cubes and the Metadata Server

When an OLAP server is defined with SAS Management Console, this information is saved in a repository within a metadata server. When cubes are defined within an OLAP server (through SAS OLAP Cube Studio or PROC OLAP), metadata about those cubes is also saved on that metadata server. When the OLAP server is loaded, it must contain connection information for that server, which includes the following:

- machine name or host name
- port number

- protocol (com or bridge)
- user ID

  Note: The password is case-sensitive and needs to be enclosed in quotation marks for lowercase or mixed-case passwords. △

- password
- repository (this is the name of the metadata repository that contains the cube metadata.)
- schema (this is the collection of cubes that are available to this server.)

Other SAS applications use their metadata to report on the cube as well.

Specifying Metadata Server Options in SAS OLAP Cube Studio

When SAS OLAP Cube Studio is executed, a window appears that enables you to select a metadata profile. You are then given the option to select an existing profile or create a new one. Select the **`Create a new metadata profile`** radio button. This opens the Metadata Profile Wizard. This wizard prompts you for a machine (host) name, a port number, a userID, and a password. This is information that is used by SAS OLAP Cube Studio to create and store metadata for the cubes that are created in this session.

Specifying Metadata Server Options When Invoking SAS OLAP Server

Metadata server options are set on the SAS command line when you execute an OLAP server:

```
../sas.exe -objectserver -nologo -noterminal -METAREPOSITORY default
-METAUSER olaptst1-METAPASS xxxxxx -METAPROTOCOL BRIDGE  -METASERVER
t1044.us.sas.com -METAPORT 9999 -objectserverparms
"multiuser classfactory=f3f46472-1e31-11d5-87c2-00c04f38f9f6
SERVER= omsobj:LogicalServer?@Name='permng''.
```

SERVER= is the name of the OLAP server as defined in SAS Management Console.

Note: For more information about metadata server administration, see the *SAS Metadata Server: Setup Guide*. △

Understanding Change Management in SAS OLAP Cube Studio

In SAS Open Metadata Architecture, *change management* is a facility that is used to implement the primary functions of metadata promotion, metadata replication, and metadata source control.

Metadata Promotion — enables you to copy the contents of a metadata repository to another repository and to specify changes in the metadata that are stored in the target repository. For example, you can use this feature to move metadata from a testing or project environment to a production environment. In such a scenario, you would change ports, hosts, and schema names because the metadata has been moved from one computing environment to another. You can preserve the integrity of the project metadata and at the same time make changes to enable the metadata to work in the production environment. Metadata

promotion must be performed in SAS Management Console. For details, see "Promoting Metadata" in the Metadata Manager help in SAS Management Console. Also see the *SAS Management Console User's Guide*.

Metadata Replication — enables you to copy the contents of a metadata repository to another repository. Replication is used to make an exact copy of a metadata repository in a new location. For example, it can be used to back up a repository. You can replicate a repository and all, some, or none of its dependent repositories. The replication task can also be saved to run at a scheduled time. Metadata replication must be performed in SAS Management Console. For details, see "Replicating Metadata" in the Metadata Manager help in SAS Management Console. Also see the *SAS Management Console User's Guide*.

Metadata Source Control — enables multiple users to work with the same metadata repository at the same time without overwriting each other's changes. SAS ETL Studio provides features for metadata source control. For details about how SAS ETL Studio supports metadata source control, see "Understanding Metadata Source Control" and "Using Metadata Source Control in SAS ETL Studio" in the help for SAS ETL Studio.

While SAS OLAP Cube Studio does not provide the features to check in or check out objects, the locked state of an object is honored in SAS OLAP Cube Studio. This enables users to simultaneously work with the same metadata repository, without overwriting each other's changes.

Accessing OLAP Cubes from SAS: SQL Pass-Through Facility for OLAP

The SQL Pass-Through facility enables a SAS user to connect to an OLAP server and execute cube queries within the PROC SQL environment. PROC SQL establishes a connection to an OLAP server by using the PROC SQL CONNECT statement.

After a connection is made to the OLAP server, multiple queries can be submitted by using the OLAP query language, Multidimensional Expressions (MDX). These queries are run against existing OLAP cubes. A PROC SQL query is then closed after all observations (rows) of data are returned.

To disconnect from the server, you must submit the PROC SQL DISCONNECT statement.

The main function of the SQL Pass-Through facility for OLAP is to query data, but MDX commands can be submitted that create named sets, globally scoped sets, drill-through paths, and other OLAP components. Additionally, named sets that are created by using MDX can then be used to run queries. It is important to note the following conditions:

- These sets are only available during the current PROC SQL session.
- Other PROC SQL sessions cannot access or reference these sets.
- After a PROC SQL connection is disconnected, any session-created sets are discarded.

Conversion Issues

OLAP cube data is multidimensional and flexible in regard to data name lengths and restrictions. However, when PROC SQL sends a query to the OLAP server, data is returned in a flattened, tabular format that contains rows (observations) and columns (variables).

The SAS OLAP Server has unique naming conventions that specify valid column names, lengths, and types. Column names that are returned from SAS OLAP can contain characters (periods, spaces, brackets) and can be unrestrained in length. Additionally, OLAP types can be variable-length strings, floating-point numbers, or integers. This differs from SAS data set naming conventions, and some conversion is necessary.

VALIDVARNAME

The SQL Pass-Through facility supports the existing SAS option VALIDVARNAME. You can specify the VALIDVARNAME option to control variable names. The current default setting for VALIDVARNAME is V7, and variable names can be a maximum of 32 characters in length. Each variable must start with a letter or the underscore character and can contain letters, underscores, and numbers. Uppercase and lowercase letters are also allowed.

When converting column names to SAS variable names, the SQL Pass-Through facility for OLAP will

- truncate the column name to the maximum size that is allowed.
- replace any invalid characters with an underscore.
- use a numeric suffix to differentiate between duplicate variable names that are generated during the data conversion.

Note: For additional information about naming restrictions for SAS OLAP Server, see "Naming Guidelines for SAS OLAP Server" on page 120. △

Note: For further information about the VALIDVARNAME= system option, see "VALIDVARNAME=Systerm Option" and "Names in the SAS Language" in the *SAS Language Reference: Dictionary*. △

Data Types

OLAP query results that contain member names or strings are converted to a fixed length CHAR type. All OLAP numeric types are converted to standard SAS numeric types (8-byte floating point). Missing values are handled by standard SAS conventions.

PROC SQL Syntax

Here is an example of the basic syntax that is used to connect to an OLAP server and execute a cube query:

```
proc sql;
  connect to dbms-name (connection options);
  execute (MDX query);
  select . . . from connection to remote | alias (dbms-query);
  disconnect from dbms-name;
  quit;
```

CONNECT TO REMOTE
establishes a connection to a remote DBMS or to remote SAS data through a SAS server. This statement is required. Remote SQL Pass-Through (RSPT) does not support implicit connection.

DBMS=dbms-name
specifies the name of the remote DBMS that you want to connect to. For SAS OLAP Server, the dbms-name is *saseolap* or *olap*.

DISCONNECT FROM REMOTE | alias
ends the connection to the remote DBMS or to the SAS SQL processor in the server SAS session.

EXECUTE (SQL-statement) BY REMOTE | alias
specifies an SQL statement to be executed by the SAS SQL processor or by the remote DBMS in the server SAS session.

SELECT . . . FROM CONNECTION TO REMOTE | alias (dbms-query);
specifies the connection to the remote SAS SQL processor or the remote DBMS as the source of data for the SELECT statement and the recipient of the dbms-query.

Here are the bridge server *connection-options*:

HOST=*machine-name*
specifies either the DNS name or the IP address of the machine that is hosting the OLAP server.

PORT=*port-number* | SERVICE=*service-name*
either the port-number or service-name is required. The *port-number* specifies the numeric value of the port on which the OLAP server resides. The *service-name* is used to look up the port number of the machine that is hosting the OLAP server.

USER=*userid*
a string that specifies the user's identification for the specified OLAP server. If included, this option is enclosed within parentheses with the required arguments and any other options.

PASS=*password*
a string that specifies the password for the user who is identified with the USER= option. If included, this option is enclosed within parentheses with the required arguments and any other options.

This is the COM server *connection-option*:

machine-name
specifies either the DNS name or the IP address of the machine that is hosting the OLAP server. This is the only argument that is used for COM server connections.

Note: For detailed information about PROC SQL syntax see "Overview of the Pass-Through Facility" in *SAS/ACCESS for Relational Databases: Reference* and "Syntax for Remote SQL Pass-Through (RSPT) Facility" in *SAS/SHARE User's Guide*. △

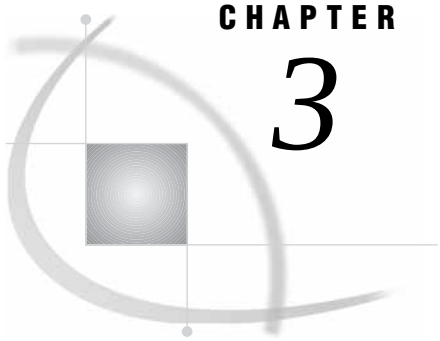
SQL Pass-Through Example

In the following example, PROC SQL connects to the pass-through facility for OLAP to create a new data set named temp, which contains all the variables that are returned from the multidimensional expression (MDX) defined in the SELECT query. The OLAP server returns query results in a tabular format known as a flattened rowset. The table rows become the observations of the output data set, and the table columns become the variables. After all the rows are returned, PROC SQL closes the query. The server

connection is terminated when the program encounters a DISCONNECT statement or when the PROC SQL step ends.

Note: Because the OLAP server does not impose the same restrictions on column names, types, and lengths that SAS imposes on data sets, some conversion might be required. △

```
100        proc sql;
101           connect to olap (host=localhost service=olap1);
102           create table temp as select * from connection to olap
103           (
104              select { dealers.dealer.members } on 0,
105                     { [cars].[Car].members,
106                       [cars].[Color].members } on 1
107              from mddbcars
108           );
109          disconnect from olap;
110          quit;
```



CHAPTER

3

Building and Updating Cubes

Background

At this point in the cube-building process, the collecting and scrubbing of the data should be finished as well as planning a dimensional design. After you have collected and analyzed your data, you are ready to create the cube. When you define the cube,

you define the dimensions and measures for the cube along with information about how aggregations should be created and stored. There are two methods of creating a cube:

- You can submit PROC OLAP code by using either the SAS Program Editor or a batch job. If you use PROC OLAP, the cube is created, and then the cube definition is stored in a metadata repository.
- You can use the Cube Designer interface in SAS OLAP Cube Studio to define and create the cube. The Cube Designer first stores the cube definition in a metadata repository, and then submits a shorter form of PROC OLAP code to create the cube.

Note: The Cube Designer can also be launched from *SAS ETL Studio*. △

Preparations for Building a Cube

To build a cube by using either PROC OLAP or SAS OLAP Cube Studio, you must complete several preliminary tasks:

- Configure a metadata server.
- Define an OLAP server in the metadata. The server does not need to be running to create cubes, but it must be defined in the metadata.
- Analyze the data to determine the location of the table(s) that will be used to build your cubes and what dimensions and measures will be created.
- Define the table(s) that will be used to create the cube in the metadata. You do this by using SAS ETL Studio or by using SAS OLAP Cube Studio and SAS Management Console as follows:
  - Use SAS Management Console to define, in the metadata, the server that will be used to access the tables. This is a SAS application server with a workspace server component.
  - Use SAS Management Console to define, in the metadata, the SAS library that contains the table.
  - In SAS OLAP Cube Studio, specify the server that will be used to access the tables. To set the server, select

    Tools ► Options

    Or, if the shortcut bar is displayed, select **Options** to set the server.
  - In SAS OLAP Cube Studio, select

    Source Designer

    to load the table definitions (or other information source) as follows:
    - From the shortcut bar, select

      Tools ► Source Designer

      or select

      Source Designer
    - Select a Source Type (SAS, ODBC, etc.), and then select **Next**.
    - If you have not specified a server, or if the server that is specified is not valid, then you will be prompted again for a server.
    - Select the SAS Library that contains the tables that you want to define, and then select **Next**.
    - Select the tables to define, and then select **Next**.
    - Select **Finish**. The table definitions are loaded into the metadata.

- If you start to create a cube and do not see the table that you need to continue, then you can select the Define Table button in any of the windows that prompt for tables.

- In the Finish window of the cube designer, you are given the option to create the physical cube. The metadata definition is always stored as you leave the Finish window. However, you can defer creation of the physical cube. If you choose to create the cube as you leave the Finish window, then you must have a SAS Workspace Server defined that you can submit PROC OLAP code to. This server is defined in SAS Management Console.

Note: For further information about the different data types that you can use to load cubes from, see "Loading Cubes" on page 121. △

Note: The SAS Metadata Server enables duplicate librefs to be defined in the metadata. To ensure that the correct SAS library definition is found on the metadata server, you should assign the libref by using the LIBNAME statement for the metadata engine before submitting the PROC OLAP code. Otherwise, PROC OLAP will select the first library definition that it finds with your specified libref, and it will associate your cube metadata with that definition. The selected library definition might or might not contain a description of the SAS data set that was actually used to build your cube. For more information about using the LIBNAME statement for the metadata engine, see "Statements" in *SAS Language Reference: Dictionary*. △

Note: When a SAS OLAP cube is created, a directory for that cube is also created. This directory is assigned the same name as the cube, but in uppercase letters. For example: When you save a cube in c:\olapcubes and name the cube Campaigns, the cube is saved in the directory c:\olapcubes\CAMPAIGNS. △

For further information about preliminary setup and configuration steps, see "Installing and Configuring SAS OLAP Server" on page 8.

Storage Location Requirements for Cube Metadata and Related Objects

When storing metadata that describes a cube, the metadata objects that describe the cube and the cube's associated libraries and source tables must be stored in the same repository, or the metadata that describes the cube must be in a custom repository that is dependent on the repository that contains the library and table objects. Otherwise, you will not be able to create the cube. In addition, the library and table objects that are referenced by a cube must always be in the same repository. The following options illustrate these conditions:

- The library, table, and cube objects can be in a Foundation repository.
- The library, table, and cube objects can be in Project A, which is dependent on the Foundation repository.
- The library and table objects can be in the Foundation repository, and the cube object can be in Project A.
- The cube object cannot be in the Foundation repository, and the library and table objects cannot be in Project A.
- The table object cannot be in the Foundation repository, and the library and cube objects cannot be in Project A.
- The library object cannot be in the Foundation repository, and the table and cube objects cannot be in Project A.

Building a Cube from a Detail Table

SAS OLAP Cube Studio

You can build an OLAP cube by using the Cube Designer in SAS OLAP Cube Studio. In this example, we are using data from a recent product marketing campaign. We want to establish measures and summaries of various aspects of our data such as geographic location of potential voters, age of potential voters, and revenue summaries. Our data is held in a detail table called olapsio.campnrml.

1 Define the metadata profile. The File menu options for New Metadata Profile and Open Metadata Profile allow you to define and connect to a metadata server. At the Open a Metadata Profile dialog box, you can choose to either create a new metadata profile or edit an existing one. Enter the machine information of the metadata server that you will connect to and retrieve a data source from.

   - At the Connection Information dialog box, enter the machine ID, port, your user ID, and password.

     Note: These options are the equivalent of the METASVR statement options:

     - HOST=
     - PORT=
     - USERID=
     - PW=.

     △

   - At the Repository Selection dialog box, select a default repository.

2 Enter general cube information.

   - After you have established a metadata server, you can begin to create a cube. Select **`Cube Designer`** from the shortcut menu. At the Cube Designer – General dialog box, enter the general cube information. For input type, you select **`Detail Table`**.
     - Cube Name
     - Description
     - Repository
     - OLAP Schema
     - Path in the file system to store the cube
     - Input Type

     Note: These options are equivalent to the PROC OLAP and METASVR statement options:

     - CUBE=
     - DESC=
     - REPOSITORY=
     - OLAP_SCHEMA=
     - PATH=.

     △

- At the Cube Designer – Input dialog box, select a data source or detail table for your cube. If one does not exist for your data, select **Define Table**, and then define the source that you will import your metadata from.

  Note: These options are equivalent to the
  - Source Designer function in the Tools Menu
  - DATA|FACT= option for PROC OLAP.

  △

- At the Cube Designer - Drill-Through dialog box, you select or define an optional drill-through table. Drill-through tables can be used by client applications to provide a view from processed data into the underlying data source.

  If a drill-through table does not exist for your data, select **Define Table**, and then define the source that you will import your metadata from.

Note: These options are equivalent to the
- Source Designer function in the Tools Menu
- DRILLTHROUGH_TABLE | DT_TABLE | DT_TBL= option for PROC OLAP.

△

The **Table Options** button that is available at both the Cube Designer - Input and the Cube Designer - Drill-Through dialog boxes, opens the **Table Options** dialog box. It enables you to specify data set options that are used to open the data set. For example, you could enter a WHERE clause or subsetting information that is then applied to the selected table when it is opened. The options are stored as part of the cube and then reapplied when the data is accessed at run time. You can also specify data set options in the Dimension Designer – General window (for use with star schemas) and the Stored Aggregates window (for use with summarized tables). For more information, see "Data Set Options" in *SAS Language Reference: Concepts*.

3 Define dimensions, levels, and hierarchies. Now that your basic metadata server and cube information has been entered, you can define the different dimensions and their respective levels and hierarchies. For this example, you will define the following dimensions and levels:

- Campaigns
  - campaign type
  - campaign
  - subcampaign.
- Campaign Dates
  - campaign start year
  - campaign start month
  - campaign start day.
- Geographic
  - division
  - region
  - client_id.
- Customer Age
  - age group1
  - age group2.
- Products

- □ product group
- □ product type.

Define the dimensions for the cube. For each dimension you will define the dimension, its levels, and its hierarchies.

- □ At the Cube Designer - Dimensions dialog box, select the **Add** button. This opens the Dimension Designer - General dialog box. Enter the following information:
  - □ dimension name
  - □ caption
  - □ description
  - □ type of dimension (standard or time)
  - □ sort order.

  Note: If you are using a star schema, enter the Dimension Tables Definition information. △
- □ Select **Dimension Levels** from the Dimension Designer - Levels dialog box.
- □ Next, define properties such as format, time type, and sort order from the Dimension Designer - Level Properties dialog box.
- □ Next, define hierarchies for the levels from the Dimension Designer - Define a Hierarchy dialog box.
- □ Repeat this process for each dimension.

Note: Use the DIMENSION, HIERARCHY, and LEVEL statements here. For time-specific levels in a dimension, the LEVEL statement is required. Also, there can be only one time-specific dimension. △

4 Define measures. You can now define the measures for the cube. In your example, you want measures for revenue totals and the number of customers you tracked during the campaigns. Define the measures for the cube at the Cube Designer - Select Measures dialog box.

Modify any measure attributes such as Measure Captions and Formats at the Cube Designer - Measure Details dialog box.

Note: The MEASURE statement is used here. △

5 Define member properties. You can now define the member properties for any needed cube members. A *member property* is an attribute of a dimension member. A member property is also an optional cube feature that is created in a dimension to provide users with additional information about members. For this example, you can define the zip code or postal code as a member property. Define member properties at the Cube Designer - Member Property dialog box.

At the Define a Member Property dialog box, enter the member property name, level, column, and caption.

Note: The PROPERTY= statement is used here. △

6 Define aggregations. You can now define the aggregations for the cube. *Aggregations* are summaries of detailed data that is stored with a cube or referred by a cube. They can help reduce the build time that is required for the cube and contribute to faster query response. Define the aggregations for the cube from the Cube Designer - Generated Aggregations dialog box.

Select the levels for the aggregation with the Specify a generated user-defined aggregation dialog box.

Note: The AGGREGATION= statement is used here. △

7 Build the cube. You can now build the cube. You can choose to build the cube and register it to the metadata repository, or you can register the cube to the metadata repository. At the Cube Designer - Finish dialog box, review the settings for the cube, and then select one of the cube creations options.

   Select whether to save the generated PROC OLAP code. At the Save PROC OLAP Code dialog box, enter the file location where you want to save the resulting code.

   Select the **Finish** button from the Cube Designer - Finish dialog box.

Note: When a SAS OLAP cube is created, a directory for that cube is also created. This directory is assigned the same name as the cube, but in uppercase letters. For example, when you save a cube in c:\olapcubes and name the cube Campaigns, the cube is saved in the directory c:\olapcubes\CAMPAIGNS. △

Saving a Cube's PROC OLAP Code

In SAS OLAP Cube Studio, you can elect to save the PROC OLAP code that is generated when a cube is built. The code is saved to a text file that you specify. The information saved in the file includes the following items:

- the SAS LIBNAME statement
- any FMTSEARCH statements
- any additional SAS code
- the PROC OLAP statement
- the METASVR statement
- all other PROC OLAP statements.

You can access the Save PROC OLAP Code window by using one of the following methods:

1 On the navigation tree in SAS OLAP Cube Studio, right-click on a cube and select **Save PROC OLAP code**.

2 In the Finish window in the Cube Designer wizard, right-click the **Save PROC OLAP Code** button.

The Save PROC OLAP Code window opens. In the Path field, enter the name and location of the file that you are saving to. An example is c:/temp/olapcode.txt.

PROC OLAP

You can build an OLAP cube with PROC OLAP and execute it in a SAS session. Running PROC OLAP registers your cube and its sources in a metadata repository. It also creates the files that make up the cube. These are the *possible* input types for an OLAP cube:

- a fact table (specified in the PROC OLAP statement DATA= option)
- dimension tables (specified in the DIMENSION statement DIMTBL= option)
- presummarized tables (specified in the AGGREGATION statement TABLE=option)

Note: You use the DATA= option when you use a detail table as the input. △

In this example, you are using data from a recent product marketing campaign. You want to establish measures and summaries of various aspects of your data such as geographic location of potential voters, age of potential voters, and revenue summaries. Our data is held in a table called olapsio.campnrml.

1 Define the metadata profile and general information. You use the PROC OLAP and METASVR statements here. The fact table is specified in the PROC OLAP

statement DATA= option. The METASRV statement is used to establish the metadata connection. It identifies the metadata repository in which existing cube metadata information exists or in which metadata about a new cube should be stored. The statement is also used to provide a user's identification and password for the identified repository. Also the DRILLTHROUGH_TABLE= option is used here to indicate the drill-through table. Drill-through tables can be used by client applications to provide a view from processed data into the underlying data source.

```
proc olap data=olapsio.campnrml cube=Campaign1 path= c:\cubes
drillthrough_table=olapsio.campnrml;


 metasvr host=&host port=&port  protocol=&protocol
userid=&userid pw=&pw repository=&repos olap_schema=&olap_serv;
```

In OLAP Cube Studio, you define the above options in the following windows:

- Metadata Profile
- Connection Information
- Cube Wizard - General
- Cube Wizard - Input.

2 Define dimensions, levels, and hierarchies. Now that your basic metadata server and cube information has been entered, you can define the different dimensions and their respective levels and hierarchies. For this example, you define the following dimensions and levels:

- Campaigns
  - campaign type
  - campaign
  - subcampaign.
- Campaign Dates
  - campaign start year
  - campaign start month
  - campaign start day
- Geographic
  - division
  - region
  - client_id.
- Customer Age
  - age group1
  - age group2.
- Products
  - product group
  - product type.

You use the DIMENSION, HIERARCHY, and LEVEL statements here.

Note: For time-specific levels in a dimension, the LEVEL statement is required. Also, there can be only one time-specific dimension. △

```
dimension campaigns hierarchies=(campaigns) caption="Campaigns"
sort_order=ascformatted;
hierarchy campaigns levels=(campaign_type campaign sub_campaign);

dimension campaign_dates hierarchies=(campaign_dates)
 caption="Campaign launch dates" type=time;
hierarchy campaign_dates levels=(campaign_start_year campaign_start_month
    campaign_start);
level campaign_start_year type=year;
level campaign_start_month type=months;
level campaign_start type=days;

dimension geographic hierarchies=(geographic) caption="Geographic";
hierarchy geographic levels=(division region client_id);

dimension customer_age hierarchies=(customer_age) caption="Customer age";
hierarchy customer_age levels=(age_group_1 age_group_2);

dimension products hierarchies=(products) caption="Products";
hierarchy products levels=(product_group product_type);
```

In OLAP Cube Studio, the above options are defined in the following windows:

- Dimension Designer - General
- Dimension Designer - Levels
- Dimension Designer - Level Properties
- Dimension Designer - Define a Hierarchy.

3 Define measures. You can now define the measures for the cube, both stored and derived. A *measure* is an input column and a roll-up rule (statistic). Only certain measures are physically stored. Other measures are derived from the stored measures at run time. In this example, you want measures for revenue totals and the number of customers you tracked during the campaigns.

You use the MEASURE statement here.

```
measure revenue_sum column=revenue stat=sum format=dollar15.2;
measure number_of_customers_sum column=number_of_customers
 stat=sum format=12.0;
```

In OLAP Cube Studio, the above options are defined in the following windows:

- Cube Designer - Select Measures
- Cube Designer - Measure Details.

4 Define member properties. You can now define the member properties for any needed cube members. A *member property* is an attribute of a dimension member. A member property is also an optional cube feature that is created in a dimension to provide users with additional information about members. For this example, you can define the zip code or postal code as a member property.

You use the PROPERTY statement here.

```
PROPERTY  zipcode-region column = post_code
HIHIERARCHY= geographic
LEVEL = region;
```

In OLAP Cube Studio, the above options are defined in the following windows:

- Cube Designer
- Member Property
- Define a Member Property.

5 Define aggregations. You can now define the aggregations for the cube. Aggregations are summaries of detailed data that is stored with a cube or referred by a cube. Their existence can reduce cube query time. If all aggregations are to be generated at the time of cube creation (MOLAP cube), then you can select specific aggregations that must be created in addition to the NWAY, which is the only aggregation that PROC OLAP makes by default.

You use the AGGREGATION statement here.

```
aggregation product_group product_type age_group_1 age_group_2
division region;
```

Note: The input type – presummarized tables – can also be specified in the AGGREGATION statement by using the TABLE= option. △

In OLAP Cube Studio, the above options are defined in the following windows:

- Cube Designer - Generate Aggregations
- Specify a generated user-defined aggregation.

6 Build the cube. You can now build the cube. Execute the PROC OLAP statement within the SAS System or in batch-mode. Here is the complete PROC OLAP code.

```
proc olap data=olapsio.campnrml cube=Campaign1 path= c:\cubes;

metasvr host=&host port=&port protocol=&protocol userid=&userid pw=&pw
repository=&repos olap_schema=&olap_serv;

dimension campaigns hierarchies=(campaigns) caption="Campaigns"
sort_order=ascformatted;
hierarchy campaigns levels=(campaign_type campaign sub_campaign);

dimension campaign_dates hierarchies=(campaign_dates)
caption="Campaign launch dates" type=time;
hierarchy campaign_dates levels=(campaign_start_year campaign_start_month
campaign_start);
level campaign_start_year type=year;
level campaign_start_month type=months;
level campaign_start type=days;

dimension geographic hierarchies=(geographic) caption="Geographic";
hierarchy geographic levels=(division region client_id);

dimension customer_age hierarchies=(customer_age) caption="Customer age";
hierarchy customer_age levels=(age_group_1 age_group_2);

dimension products hierarchies=(products) caption="Products";
hierarchy products levels=(product_group product_type);

measure revenue_sum column=revenue stat=sum format=dollar15.2;
measure number_of_customers_sum column=number_of_customers
stat=sum format=12.0;
```

```
aggregation product_group product_type age_group_1 age_group_2
division region;

run;
```

Note: Any libraries or fmtsearch options must be specified prior to running the PROC OLAP code. △

Note: When a SAS OLAP cube is created, a directory for that cube is also created. This directory is assigned the same name as the cube, but in uppercase letters. For example, when you save a cube in c:\olapcubes and name the cube Campaigns, the cube is saved in the directory c:\olapcubes\CAMPAIGNS. △

Building a Cube from a Summary Table

In this example, you can build a cube with fully summarized data. A *summary table* is a data source that contains a crossing of all dimensions for a cube. In this example, the data for the Campaign cube has been summarized into a table called CAMPAIGN_SUMMARY. The table contains a column

- for each of the levels that you want
- for each of the stored measures, total revenue, and total number of customers
- for the member property and zip code region

The table contains only the NWAY. The data set was produced with the following SAS code:

```
proc summary data=olaplib.campaign nway noprint;
class campaign_type campaign sub_campaign campaign_start_year
campaign_start_month campaign_start division region client_id
age_group_1 age_group_2 product_group product_type;
var revenue number_of_customers;
id post_code;
output out=olaplib.campaign_summary sum=totrev totcust;
run;
```

To create the cube by using SAS Cube Studio, follow these steps:

1. Start SAS Cube Studio and connect to the appropriate metadata server.
2. Define the campaign_summary table in the metadata by using the Source Designer.
3. To create a cube, select **Cube Designer** from the shortcut menu.
4. At the Cube Designer - General dialog box, enter the following information:
   - Cube Name
   - Description
   - Repository
   - OLAP Schema
   - Path in file system to store the cube
   - Input Type.

   For input type, select **Fully Summarized Table**.
5. At the Cube Designer - Input dialog box, select a data source or detail table for your cube. For this example, select the **campaign_summary** table.

   Note: If a detail table does not exist for your data, then select **Define Table** and define the source that you will import your metadata from. △

Note: When you create a cube from a detail table or star schema, it is the equivalent of specifying the DATA= or FACT= options in the PROC OLAP statement. When you select Fully Summarized Table, this is the same as not specifying DATA= or FACT= and then specifying an AGGREGATION statement that has all levels listed and has the TABLE= option specified with the table name. Here is an example:

```
aggregation campaign campaign_type?..   /
TABLE=olaplib.campaign_summary name='Default';
```

△

6 On the Cube Designer - Drill-Through panel, select whether or not you will have a drill-through table. In most cases, when you use a fully summarized table to load the cube you specify as the drill-through table, the table that the summarized table was created from. In this example, that is the campaign table. Select the radio button **Select drill-through table from list**, and then select the **Campaign** table.

7 Now that your basic metadata server and cube information has been entered, you can define the different dimensions and their respective levels and hierarchies. This example cube has these dimensions:

- Campaigns
  - campaign type
  - campaign
  - subcampaign.
- Campaign Dates
  - campaign start year
  - campaign start month
  - campaign start day.
- Geographic
  - division
  - region
  - client_id.
- Customer Age
  - age group1
  - age group2.
- Products
  - product group
  - product type.

Define the dimensions for the cube. For each dimension, define the dimension, its levels, and its hierarchies.

- At the Cube Designer - Dimensions dialog box, select the **Add** button. This opens the Dimension Designer - General dialog box. Enter the following information:
  - dimension name
  - caption

- □ description
- □ type of dimension (standard or time)
- □ sort order.

- □ Select the necessary dimension levels at the Dimension Designer - Levels dialog box.
- □ Define properties such as format, time type, and sort order at the Dimension Designer - Level Properties dialog box.
- □ Define hierarchies for the levels at the Dimension Designer - Define a Hierarchy dialog box.
- □ Repeat this process for each dimension.

Note: You use the DIMENSION, HIERARCHY, and LEVEL statements here. For time-specific levels in a dimension, the LEVEL statement is required. Also, there can only be one time-specific dimension. △

8 You can now select the stored (base) measures for the cube in the Cube Designer - Select Stored Measures window. When loading from a detail table, the base and derived measures are generated from a single column in the detail table. For example, a detail table that has a column ACTUAL can have two measures – ACTUAL_SUM and ACTUAL_AVG – that are created from the column. However, with a fully summarized table, you must have one column for any base measure that you want to include in the cube. The base statistics are SUM, N, NMISS, MIN, MAX, and USS. Measures that are created with these statistics must have a single column in the summarized table. For this example, we have two base or stored measures: TOTREV and TOTCUST. Select the two columns in this window.

Note: This step is equivalent to using the AGGR_COLUMN option in the MEASURE statement. △

9 In the Cube Designer - Assign Stored Measures window, you can specify the Statistic and Analysis Group options for the stored statistics. Select **SUM** as the statistic for both the TOTREV and TOTCUST measures. For Analysis group, specify REVENUE for TOTREV and NUMBER_OF_CUSTOMERS for TOTCUST. If the table contained two measures from the same analysis column, both of the base measures would have the same analysis group specified. For example, if campaign_summary contained a column called REVN, which is the number of non-missing values for the REVENUE column, then we could have a base measure REVN with the statistic of count (N) and an analysis group of REVENUE.

Note: The analysis group is equivalent to the COLUMN|ANALYSIS option in the MEASURE statement. △

10 In the Cube Designer – Select Derived Measures window, specify any measures that will be derived from the base or stored measures. Each derived measure is based on a set of required stored measures. If the stored measures for an analysis group do not include all those required for a specific derived measure, then that measure cannot be included in the cube. For example, TOTREV is the SUM of the REVENUE group. You cannot include AVGREV because you do not have the N or count of the REVENUE group in the stored measures.

11 In the Cube Designer - Edit Measure Details window, specify captions, formats, and other information about the measures that are listed.

12 Define the member properties for any needed cube members. A member property is an attribute of a dimension member. A member property is also an optional cube feature that is created in a dimension to provide users with additional information about members. Define member properties at the Cube Designer - Member Property dialog box.

At the Define a Member Property dialog box, enter the member property name, level, column, and caption.

Note: You use the PROPERTY= statement here. △

13 Define the aggregations for the cube. Aggregations are summaries of detailed data that is stored with a cube or referred by a cube. They can help reduce the build time that is required for the cube and contribute to faster query response. If you have additional aggregation tables, select them in the Cube Designer - Aggregation Tables window.

14 Define the stored aggregations in the Cube Designer - Stored Aggregations window. These are aggregations that are contained in the additional input tables that were selected in the Cube Designer - Aggregation Tables window.

Note: Defining aggregations in this panel is equivalent to using the AGGREGATION statement with the TABLE= option. △

15 In the Cube Designer - Generated Aggregations window, define additional aggregations that will be generated when the cube is built.

16 Build the cube. In the Cube Designer - Finish window, you can select whether or not you want the cube to be physically created after the metadata is saved. When you select the **Finish** button, the metadata for the cube is always saved. If you select **Save the metadata and create the cube**, the short form of the PROC OLAP code is generated along with the necessary LIBNAME statements and submitted to an application server. You can also select whether to save the PROC OLAP code that is generated. At the Save PROC OLAP Code dialog box, enter the file location where you want to save the resulting code.

Note: When a SAS OLAP cube is created, a directory for that cube is also created. This directory is assigned the same name as the cube, but in uppercase letters. For example, when you save a cube in c:\olapcubes and name the cube Campaigns, the cube is saved in the directory c:\olapcubes\CAMPAIGNS. △

Here is the complete PROC OLAP code:

```
 PROC OLAP   DrillThrough_Table=olaplib.CAMPAIGN_SUMMARY
          cube="Summary1"
          Path="c:\cubes"
          Description="Summary1"
         ;
 METASVR host=localhost port=9999 protocol=bridge userid=userid pw=pw
repository=Foundation olap_schema= ''OLAP Schema'';

 DIMENSION Campaigns hierarchies=(Campaigns )
          CAPTION='Campaigns'
          SORT_ORDER=ASCENDING ;

          HIERARCHY Campaigns levels=( Campaign_Type Campaign Sub_Campaign )
          CAPTION='Campaigns'
          DEFAULT
          ;

          LEVEL Campaign_Type
          CAPTION='Campaign Type'
          SORT_ORDER=ASCENDING
          ;
```

```
        LEVEL Campaign
        CAPTION='Campaign ID'
        SORT_ORDER=ASCENDING
        ;

        LEVEL Sub_Campaign
        CAPTION='Sub Campaign'
        SORT_ORDER=ASCENDING
        ;


DIMENSION CampaignDates hierarchies=(CampaignDates )
        CAPTION='CampaignDates'
        TYPE=TIME SORT_ORDER=ASCENDING ;

        HIERARCHY CampaignDates levels=( Campaign_Start_Year
                          Campaign_Start_Month Campaign_Start )
        CAPTION='CampaignDates1'
        DEFAULT
        ;

        LEVEL Campaign_Start_Year TYPE=YEAR
        CAPTION='Campaign Start Year'
        SORT_ORDER=ASCENDING
        ;

        LEVEL Campaign_Start_Month TYPE=MONTHS
        CAPTION='Campaign Start Month'
        SORT_ORDER=ASCENDING
        ;

        LEVEL Campaign_Start TYPE=DAYS
        CAPTION='Campaign Start'
        SORT_ORDER=ASCENDING
        ;


DIMENSION Geographic hierarchies=(Geographic )
        CAPTION='Geographic'
        SORT_ORDER=ASCENDING ;

        HIERARCHY Geographic levels=( Division Region Client_ID )
        CAPTION='Geographic'
        DEFAULT
        ;


        LEVEL Division
        CAPTION='Division'
        SORT_ORDER=ASCENDING
        ;

        LEVEL Region
        CAPTION='IFA Region'
```

```
          SORT_ORDER=ASCENDING
          ;

          LEVEL Client_ID
          CAPTION='Client id'
          SORT_ORDER=ASCENDING
          ;


DIMENSION CustomerAge hierarchies=(CustomerAge )
          CAPTION='CustomerAge'
          SORT_ORDER=ASCENDING ;

          HIERARCHY CustomerAge levels=( Age_Group_1 Age_Group_2 )
          CAPTION='AgeGroup1'
          DEFAULT
          ;

          LEVEL Age_Group_1
          CAPTION='Age Group 1'
          SORT_ORDER=ASCENDING
          ;

          LEVEL Age_Group_2
          CAPTION='Age Group 2'
          SORT_ORDER=ASCENDING
          ;


DIMENSION Products hierarchies=(Products )
          CAPTION='Products'
          SORT_ORDER=ASCENDING ;

          HIERARCHY Products levels=( Product_Group Product_Type )
          CAPTION='Products1'
          DEFAULT
          ;

          LEVEL Product_Group
          CAPTION='Product Group'
          SORT_ORDER=ASCENDING
          ;

          LEVEL Product_Type
          CAPTION='Product Code'
          SORT_ORDER=ASCENDING
          ;

MEASURE totrevSUM
          STAT=SUM
          ANALYSIS=Revenue
          AGGR_COLUMN=totrev
          CAPTION='Sum of Revenue'
          FORMAT=BEST12.
```

```
        DEFAULT
        ;

MEASURE totcustSUM
        STAT=SUM
        ANALYSIS=Number_Of_Customers
        AGGR_COLUMN=totcust
        CAPTION='Sum of Number_Of_Customers'
        FORMAT=BEST12.
        ;

RUN;
```

Building a Cube from a Star Schema

In this example, you can build a cube from a star schema. A *star schema* is a data source that contains tables in a database in which a single fact table is connected to multiple dimension tables. In this example, an international retail company sells sports and outdoor products.

To create the cube by using SAS Cube Studio, complete these steps:

1. Start SAS OLAP Cube Studio and connect to the appropriate metadata server.
2. To begin creating a cube, select **Cube Designer** from the shortcut menu.
3. In the Cube Designer - General window, enter the following information:
   - Cube Name
   - Description
   - Repository
   - OLAP Schema
   - Path in file system to store the cube
   - Input Type.

   For input type, select **Star Schema**.
4. In the Cube Designer - Input window, select a data source or detail table for your cube. For this example select the **ORDER_FACT** table. If a detail table does not exist for your data, select **Define Table**, and then define the source that you will import your metadata from.

   Note: When you build the dimensions using PROC OLAP, and if you use a fact table from a star schema, use the FACT= option. Additionally, use FACT= option to read a star schema, then use the DIMTBL= option to specify the dimension tables. △

   Note: If the cube is built from a star schema, then the keys that link the dimension table and the fact table are also defined by using the DIMKEY= and FACTKEY= options. See "DIMENSION Statement" on page 99 for further information. △
5. In the Cube Designer - Drill-Through window, determine whether or not you will have a drill-through table. In this example, you will not use a drill-through table, so you can select the option **No table for Drill-Through**.
6. In the Cube Designer - Dimension Tables window, select dimension tables that are associated with the ORDER_FACT star schema that you specified as the data source for the cube. For this example, select the following tables:

- CUSTOMER_DIM
- GEOGRAPHY_DIM
- ORGANIZATION_DIM_MOD_LEVELLED
- TIME_DIM.

7 Now that your basic metadata server and cube information has been entered, define the different dimensions and their respective levels and hierarchies. This example cube has these dimensions and levels:

- Time
  - Year_ID
  - Quarter
  - Month_Name
  - Week_Name
  - Date_ID.

  For the Time dimension, the following star schema information is also included:

Table	TIME_DIM
Key	Date_ID
Fact Key	Order_Date

- Customers
  - Customer_Name
  - Customer_Age
  - Customer_Gender
  - Customer_Group
  - Customer_Type.

  For the Customers dimension, the following star schema information is also included:

Table	CUSTOMER_DIM
Key	Customer_Id
Fact Key	Customer_Id

- Geography
  - Continent_Name
  - Country
  - State
  - Region
  - Province
  - County
  - City.

For the Geography dimension, the following star schema information is also included:

Table	GEOGRAPHY_DIM
Key	Street_Id
Fact Key	Street_Id

- Organization
  - Employee_Name
  - Job_Title
  - Salary
  - Gender
  - Company
  - Department
  - Org_Group
  - Section.

For the Organization dimension, the following star schema information is also included:

Table	ORGANIZATION_DIM
Key	Employee_Id
Fact Key	Employee_Id

Define the dimensions for the cube. For each dimension, you define the dimension, its levels, and its hierarchies.

- At the Cube Designer - Dimensions dialog box, select the **Add** button. This opens the Dimension Designer - General dialog box. Enter the following information:
  - dimension name
  - caption
  - description
  - type of dimension (standard or time)
  - sort order.

  When you define the dimensions for a cube based on a star schema, you will need to provide additional information about the dimensions in the Dimension Designer - General window. On the Star Schema Dimension Tables Definition panel, enter the following information:
  - Table
  - Key
  - Fact Key
  - Data Set Options.
- Select the necessary dimension levels at the Dimension Designer - Levels dialog box.

- Define properties such as format, time type, and sort order at the Dimension Designer - Level Properties dialog box.
- Define hierarchies for the levels at the Dimension Designer - Define a Hierarchy dialog box.
- Repeat this process for each dimension.

Note: You use the DIMENSION, HIERARCHY, and LEVEL statements here. For time-specific levels in a dimension, the LEVEL statement is required. Also, there can be only one time-specific dimension. △

8 Specify the columns or measures for the cube. In the Cube Designer - Selected Measures window, select the following columns and associated Sum statistics:

- Total_Retail_Price /Sum
- Quantity /Sum
- CostPrice_Per_Unit /Sum
- Discount /Sum.

9 Specify detail information for the measures. In the Cube Designer - Measure Details window, enter any necessary information for the different measures:

- Caption
- Format
- Unit
- Description.

For the measure Total_Retail_Price, enter a format value of DOLLAR12.2. For the measure CostPrice_Per_Unit, enter a format value of DOLLAR10.2.

10 Specify member property information for the levels in the cube. In the Cube Designer - Member Property window, select the **Add** button to create a new member property. In the Define a Member Property window, enter the following information about the member property:

- Name
- Level
- Column
- Format
- Caption Description
- Selected Hierarchies.

In this example, the following member properties are created:

Property Name	Level	Column	Caption	Selected Hierarchy
WeekDay_Number_US	date	weekday_no	US WeekDay Number	
WeekDay_Number_EU	date	weekday_eu	EU WeekDay Number	
Week_Number_EU	week_name	week_no	EU Week Number	YWD
Month_Number	month_name	month_no	Month Number	YMD
Month_Number	month_name	month_no	Month Number	YQMD
Holiday_US	date	Holiday_US	US Holidays	

11 Specify the aggregations for the cube. Aggregations are summaries of detailed data that is stored with a cube or referred by a cube. They can help reduce the build time that is required for the cube and contribute to faster query response. In the Cube Designer - Generated Aggregations window, select the **Add** button to specify aggregations and associated levels. Order the levels for the aggregations to follow the hierarchy drill path. The aggregations include

- RegionalCustomerUse
- QuarterlyCustomerUse
- YearlyCustomerUse
- WorldwideStaff
- WorldwideSalaries.

Note: When you create cubes in SAS OLAP Cube Studio, a default aggregation,which is the NWAY aggregation, is automatically created and listed in the Cube Designer - Generated Aggregations window. △

12 Build the cube. In the Cube Designer - Finish window, select whether or not you want the cube to be physically created after the metadata is saved. When you click **Finish**, the metadata for the cube is always saved. If you select **Save the metadata and create the cube**, the short form of the PROC OLAP code is generated along with the necessary LIBNAME statements and submitted to an application server. You can also select whether to save the PROC OLAP code that is generated. At the Save PROC OLAP Code dialog box, enter the file location where you want to save the resulting code.

Note: When a SAS OLAP cube is created, a directory for that cube is also created. This directory is assigned the same name as the cube, but in uppercase letters. For example, when you save a cube in c:\olapcubes and name the cube Campaigns, the cube is saved in the directory c:\olapcubes\CAMPAIGNS. △

Here is the complete PROC OLAP code:

```
proc olap cube=Star
          path=c:\cubes
          fact=olapsio.ordfact
metasvr host=localhost
         port=9999t
         protocol=bridge
         userid=userid
         pw=pw
         repository=Foundation
         olap_schema=OLAP Schema
         ;

dimension Time hierarchies=(YWD YMD YQMD) type=time
           dimtbl=olapsio.timedim dimkey=date_ID factkey=order_date
           ;
hierarchy YWD caption="Year-Week-Day"
           levels=(Year_ID Week_Name Date_ID );
hierarchy YMD caption="Year-Month-Day"
           levels=(Year_ID Month_Name Date_ID);
hierarchy YQMD caption="Year-Quarter-Month-Day"
           levels=(Year_ID Quarter Month_name Date_ID);
level year_ID       type=year;
level quarter       type=quarters;
level month_name    type=months;
level week_name     type=weeks;
```

```
level date_ID       type=days;
property WeekDay_Number_US  caption="US WeekDay Number" column=weekday_no
level=date;
property WeekDay_Number_EU  caption="EU WeekDay Number" column=weekday_eu
level=date;
property Week_Number_EU     caption="EU Week Number"    column=week_no
hierarchy=YWD   level=week_name;
property Month_Number       caption="Month Number"      column=month_no
hierarchy=YMD   level=month_name;
property Month_Number       caption="Month Number"      column=month_no
hierarchy=YQMD  level=month_name;
property Holidays_US        caption="US Holidays"       column=Holiday_us
level=date;

dimension Customers hierarchies=(PersonalData CompanyUsage)
           dimtbl=olapsio.custdim dimkey=customer_id factkey=customer_id;

hierarchy PersonalData levels=(Customer_Name Customer_Age Customer_Gender);

hierarchy CompanyUsage
           empty_char=_missing_
           levels=(Customer_Group Customer_Type);

dimension Geography hierarchies=(Geography)
          dimtbl=olapsio.geogdim dimkey=street_id factkey=street_id;
hierarchy Geography
          empty_char=_missing_
          levels=(Continent_Name Country State Region Province County City)
          ;
dimension Organization hierarchies=(PersonalStats Organization)
          dimtbl=olapsio.orgdim dimkey=employee_id factkey=employee_id;
 hierarchy PersonalStats levels=(Employee_name Job_Title Salary Gender);

 hierarchy Organization
          empty_char=_missing_
          levels=(Company Department Org_Group Section Job_Title);

MEASURE DiscountSUM STAT=SUM COLUMN=Discount;

MEASURE CostPrice_Per_UnitSUM STAT=SUM COLUMN=CostPrice_Per_Unit
          FORMAT=DOLLAR10.2
          ;
MEASURE QuantitySUM STAT=SUM COLUMN=Quantity
          CAPTION='Sum of Quantity'
          ;
MEASURE Total_Retail_PriceSUM STAT=SUM COLUMN=Total_Retail_Price
          FORMAT=DOLLAR12.2
          ;

AGGREGATION Continent_Name Country State Region Customer_Group Customer_Type
             / NAME='RegionalCustomerUse'
             ;
AGGREGATION Year Quarter Customer_Group Customer_Type
             / NAME='QuarterlyCustomerUse'
             ;
```

```
AGGREGATION Year Customer_Group Customer_Type
              / NAME='YearlyCustomerUse'
              ;
AGGREGATION Continent_Name Country State Region Province Company Department
               Org_Group Section
              / NAME='WorldwideStaff'
              ;
AGGREGATION Continent_Name Country State Region Province Employee_Name
              Job_Title Salary
              / NAME='WorldwideSalaries'
              ;
run;
```

Updating a Cube

A cube can be updated after its initial creation in order to optimize cube performance and to add or remove aggregations. Any cube update changes elements of both the physical cube and its metadata registration. You can update a cube in either SAS OLAP Cube Studio or with PROC OLAP code.

- To update an existing cube in SAS OLAP Cube Studio, select the cube that you want to modify, right-click, and select the **Edit Cube Structure** option. You can also access this function from the Actions menu. This opens the Cube Designer Wizard. Make any necessary changes to the cube. At the Cube Designer - Finish dialog box, review the settings for the cube and select one of the cube creations options. SAS OLAP Cube Studio deletes the cube, and then rebuilds it with the changes that you entered.
- To update a cube by using PROC OLAP code, modify the code as needed and save the updated code. Use the DELETE option, the DELETE_PHYSICAL option, or both from the PROC OLAP statement to delete the cube. Resubmit the modified code to rebuild the cube. See the DELETE and DELETE_PHYSICAL options for the "PROC OLAP Statement" on page 92 for further information. Here are three possible usage scenarios:

  If you are changing captions and descriptions or the dimensions or measures,
  You must use DELETE to remove the physical cube and the metadata registration. This is because you are submitting the full PROC OLAP syntax with changes.

  If the input table has new data and you only want to refresh the cube,
  You should use DELETE_PHYSICAL to delete only the physical cube. You can then submit the shorter form of the PROC OLAP syntax with only the PROC OLAP statement and METASVR statement.

  If you are optimizing cube performance by adding or deleting aggregations,
  You should not use the DELETE or DELETE_PHYSICAL options. This is because you are updating the cube in its current state.

Refreshing Cube Metadata

You can refresh the metadata for calculated members and named sets that are associated with a cube by using the Refresh Cubes function, which is available from the

SAS OLAP Cube Studio toolbar. The refresh cubes function reads the information about calculated members and named sets that is stored on the metadata server. The refresh cubes function then updates the OLAP server metadata for the cube. You must be connected to a server and have administrative permissions in order to select the Refresh Cubes function.

With the Refresh Cubes function, you can select from a list of one or more cubes to refresh. You can also select a check box that selects all cubes. The cubes listed are those cubes that are assigned to the current OLAP schema and that physically exist. When you have selected the cubes that you want to refresh, select **OK**. The refresh command is then sent to all cubes that were selected.

After the selected cubes have been refreshed, you are prompted to check other servers that the current OLAP schema is associated with.

MDX DDL REFRESH Statement

The REFRESH statement can be sent manually. You can send the REFRESH statement for each additional server that the schema is associated with.

```
REFRESH CUBE (cubename | "_ALL_" )
```

Where cubename specifies a single cube to refresh for the current server connection. Or _ALL_ specifies that all cubes are refreshed for the current server connection. Here are some examples.

This example uses the REFRESH statement to refresh the metadata associated with a STAR cube.

```
REFRESH CUBE [STAR]
```

This example uses the REFRESH statement to refresh the metadata for all cubes managed by the connected server.

```
REFRESH CUBE _ALL_
```

You can use the OLAP MDX SQL Pass-Through facility to send the DLL REFRESH statement to a server. Here is an example.

```
proc sql noerrorstop;
  connect to olap (&olapcon);
  execute
  (
    REFRESH CUBE [STAR]
  ) by olap;

proc sql noerrorstop;
  connect to olap (&olapcon);
  execute
  (
    REFRESH CUBE _ALL_
  ) by olap;
```

Defining Member Properties

When you create a SAS OLAP cube, the information that is relevant to the cube is defined with the cube hierarchy, measures, and aggregations (summaries) that will be stored with the cube. Additional information that is part of the cube member data can be included in the cube definition as a member property.

Member properties are attributes of dimension members that provide an additional gradation of information to users of the cube data. Member property information is usually not as significant as the levels and members within a dimension, and therefore, does not qualify as a level or member. However, it often has additional analytical value that can be useful at query time.

A member property is assigned to a level within a hierarchy, and a level can have multiple properties that are assigned to it. For hierarchy placement, a member property is assigned (by default) to all hierarchies that the select level is in. However, you can remove one or more (but not all) of the hierarchies that the member property is assigned to.

When you create a member property, you must specify the name, column, and level. Member property names can be shared across a cube but must be unique for a specific level within a specific hierarchy. You can also specify a caption, description, and format. The format that you specified here will be used instead of the format in the data set.

Property Statement

The PROPERTY statement is used with the PROC OLAP statement when you define a cube:

```
PROPERTY  zipcode-region  column = post_code
HIERARCHY= geographic
LEVEL = region;
```

Cube Designer

You can also establish member properties with the Member Property dialog box that is part of the Cube Designer interface in SAS OLAP Cube Studio. This is accessed after measures are defined, when you create a cube or edit a cube. Select **Add** to create a member property. At the Define a Member Property dialog box, enter the member property name, level, column, and caption.

Defining Multiple Hierarchies for a Dimension

SAS OLAP cubes are organized into dimensions and levels of data. The levels are then arranged into hierarchies. After an initial hierarchy has been created, you can define additional hierarchies for a single dimension of a cube. This enables you to have multiple possible drill paths of the same data. When you create more than one hierarchy for a dimension, the levels have some restrictions:

- A level in a dimension might be used in more than one hierarchy within that dimension. However, levels cannot be used in hierarchies that are not defined within the dimension that the level is defined in.
- Each level must be used in at least one hierarchy.
- Levels from the same dimension that are picked for an aggregation must be in the drill order for at least one hierarchy in that dimension.

You can arrange the levels in a hierarchy in any order. The one exception to this is the Time dimension. Levels in hierarchies in the Time dimension must follow a prescribed order that is determined by the numerical value that is assigned to the type.

This order is from the smallest value (Years, 16) to the greatest value (Seconds, 3,096). The dimension hierarchies also have some restrictions:

- The first hierarchy that is defined for the dimension is designated as the default. When there are multiple hierarchies, you can designate the default hierarchy for the dimension.
- Hierarchy names must be unique across the cube. If there is a single hierarchy for a dimension, then its name must be the name of the dimension Also, dimension and hierarchy names cannot be the same as a level name within that dimension.
- For any cube loaded with a star schema, in which a dimension table represents multiple hierarchies for that dimension, the dimension key that is used to join the dimension table to the fact table will be used for all hierarchies of that dimension.

Hierarchies Statement

The HIERARCHY statement is used with the PROC OLAP statement when you define a cube:

```
hierarchy campaigns levels=(campaign_type campaign sub_campaign);
```

Cube Designer

You can establish multiple hierarchies by using the Cube Designer - Dimensions window, which is located in the SAS OLAP Cube Studio Cube Designer. To add a hierarchy to an existing dimension, select a dimension, and then click **Modify**. This opens the Dimension Designer - General window. It is populated with the values for the selected dimension. Select **Next** until you reach the Dimension Designer - Hierarchy window. Select **Add** to create an additional hierarchy.

Note: You can modify existing hierarchies by selecting a hierarchy and clicking **Modify**. You can also assign a default hierarchy by selecting a hierarchy and clicking **Default**. The first hierarchy is automatically the default hierarchy. △

Note: An exception to defining multiple hierarchies for a dimension is the Time dimension. Levels in hierarchies in the Time dimension must follow a prescribed order that is determined by the numeric value that is assigned to the type. This order is from the smallest value (Year, 16) to the greatest value (Seconds, 3,096). △

In the Dimension Designer - Define a Hierarchy window, you can define a new hierarchy and select the different levels and their order for the hierarchy.

Defining Ragged Hierarchies for a Dimension

Dimension levels are arranged in one or more hierarchies. Hierarchies, by process of ordering, have a branching arrangement, and the different member levels have parent and child relationships. For instance, at company X the sales staff are located in different regions and cities in different countries. A balanced hierarchy might look like this:

- Global sales president (top of hierarchy)
- Sales presidents (per country)
- Regional sales managers
- City sales managers.

Because of differences in the cube data, hierarchies are often not balanced and possibly have missing members. For example, some sales regions might not have sales managers assigned to a specific city. Or, some countries might not have sales regions, just cities. These real-world scenarios would create hierarchies that have missing member data and possibly ragged hierarchies. This affects the drillpath of the cube data.

How Ragged Hierarchies Affect Querying

Unbalanced levels and missing hierarchy members affect the path through your cube. For example, you can descend to missing members, which means that

- you have reached the leaf member of that particular branch of the hierarchy
- the leaf member is at a higher level than the hierarchy's other leaf members.

You can also drill to missing members within a path and continue to drill down to members that are present.

Defining Ragged Hierarchies in SAS OLAP Cube Studio

The Cube Designer in SAS OLAP Cube Studio enables you to specify the missing members for a hierarchy and the type of data that is missing. Here are the Cube Designer windows that enable you to specify missing member information:

Ragged Hierarchies

Located in the Cube Designer General - Advanced Cube Options window, this tab enables s you to specify character and numeric missing member information. By default, no missing member information is indicated with the value **None**.

Dimension Designer - Level Properties

- **Ragged - Ignore Missing Members** specifies whether to ignore or use global or hierarchy-specific ragged hierarchy settings. To ignore settings, set this property to **True**. To use the settings, set this property to **False**. By default, this is set to **False**.
- **Ragged - Designate Missing Members** specifies that the Cube Designer use the specified string to identify missing values and override any global or hierarchy-specific ragged hierarchy settings. You can use up to 256 characters. The value of the True/False setting in Ragged - Ignore Missing Members controls whether or not you override any global or hierarchy-specific ragged hierarchy settings.

Dimension Designer - Define a Hierarchy

You can select one of these options from the Ragged Hierarchies tab:

- **Ignore**. From this drop-down list, select **True** to ignore the global missing member settings that you entered at the Advanced Cube Options dialog box. Select **False** to use the global settings for the current hierarchy.
- **Character**. For this hierarchy only, enter a maximum of 256 characters that will be used to identify missing character members.
- **Numeric**. For this hierarchy only, enter a maximum of 256 characters that will be used to identify missing numeric members.

PROC OLAP Options for Ragged Hierarchies

You can specify missing member information in a PROC OLAP statement with the following options:

EMPTY_CHAR= 'string'
: for ragged hierarchies, this option specifies the string that identifies missing hierarchy members. For example, by default SAS identifies missing characters as a blank space. To identify blank spaces as missing hierarchy members, enter ('') as the value of the EMPTY_CHAR= option. If you enclose the string within quotation marks, then you can use a maximum of 256 characters. If you do not use quotation marks, then the string must be a valid SAS name.

  This option can be specified in the PROC statement or in the HIERARCHY statement only. If the option is specified in the HIERARCHY statement, then it overrides the value that is specified in the PROC statement. It cannot be used in the LEVEL statement.

EMPTY_NUM= 'string'
: for ragged hierarchies, this option specifies the string that identifies missing hierarchy members whose underlying input data for the member is numeric-based. For example, by default SAS identifies missing numeric values as a single period (.). To identify periods as missing values, enter a period ('.') as the value of the EMPTY_NUM= option. If you enclose the string within quotation marks, then you can use a maximum of 256 characters. If you do not use quotation marks, then the string must be a valid SAS name.

  This option can be specified in the PROC statement or in the HIERARCHY statement only. If the option is specified in the HIERARCHY statement, then it overrides the value that is specified in the PROC statement. It cannot be used in the LEVEL statement.

  Note: This option is ignored if the IGNORE_EMPTY option is specified in the HIERARCHY statement or the corresponding LEVEL statement. △

EMPTY='string'
: specifies that PROC OLAP override any ragged hierarchy option settings in the PROC OLAP or HIERARCHY statements and instead use the specified string to identify missing hierarchy members. For example, if you set EMPTY='absent', then hierarchy members that have the value 'absent' are treated as missing for all levels in which the hierarchy appears. If you enclose the string within quotation marks, then you can use a maximum of 256 characters. If you do not use quotation marks, then the string must be a valid SAS name.

  This option can be specified only in the LEVEL statement. If specified, it overrides any values that are entered in either the PROC or HIERARCHY statements. It cannot be used in the PROC or HIERARCHY statement.

IGNORE_EMPTY
: when this option is set, then ragged hierarchy option settings in PROC OLAP or HIERARCHY statements are ignored for this level. This option can be specified in the HIERARCHY statement to ignore the PROC statement specification or in the LEVEL statement to ignore the PROC and HIERARCHY specifications.

Note: You can specify the level or hierarchy options without first specifying the global option. △

Ragged Hierarchies and Unique Member Names

In a ragged hierarchy, the parent of a member might not be at the level directly above that member. Furthermore, not all children of a member are necessarily at the same level. This can lead to a situation where two children have the same unique name.

For example, in a geography hierarchy you might have the levels state, county, and city. The state Washington might have a child at the county level called *Olympia* and another child at the city level, also named *Olympia*. The city member is not a descendant of the county member of the same name. It is a child of Washington.

In a ragged hierarchy, levels can have an unconventional structure, and unpopulated levels are not assigned a token or placeholder. As a result, the unique name for the county member is **`Geography, [All Geography], Washington, Olympia`**, and the unique name for the city member is **`Geography, [All Geography], Washington, Olympia`**.

The result of this anomaly is that the city member cannot be asked for by a unique name in a query, either through MDX or an OLE DB for OLAP (ODBO) request for metadata. It will be returned in any set that contains it so the data that is associated with it is not lost. The same applies to the children of a member such as *Olympia*. Because the server searches through the hierarchy to validate member names, a request by name for a child of *Olympia* the city will result in a bad member name error. This is because the server actually searches under the county *Olympia*.

This situation occurs only when two members with the same name share a parent. Any number of *Olympia(s)*could exist under other parents with no unusual results.

Manually Tuning Cube Aggregates

When a cube is created, aggregations can be specified by the user. Aggregations are usually created to improve query performance. After a cube is created and queries are run against the cube, users might discover that certain aggregations are not being used, and adjustments and changes to the aggregations are needed. You might want to change the levels in an aggregation, add another aggregation, or entirely remove an aggregation. The Manual Tuning function allows you to adjust and improve an existing cube by adding, dropping, or modifying aggregations.

Manual Tuning requires an active IOM server connection. When you select Manual Tuning you are prompted for a SAS IOM Server. If no valid server is available, the Manual Tuning window will not open. In addition, only MOLAP cubes that have an existing physical cube, along with a SAS Metadata Repository registration, can use the Manual Tuning function.

You can access the Manual Tuning function within SAS OLAP Cube Studio by selecting an existing cube and right-clicking to display the menu that lists the Manual Tuning function. Or, you can also access it from the main OLAP Cube Studio menu under the Actions menu. The Manual Tuning window lists the defined aggregations for the selected cube. When you finish making selections and select **`OK`**, the PROC OLAP statements are submitted to the IOM application server and the cube data is updated. With the Manual Tuning Window you can

- Add an aggregation - select **`Add`** to create a new aggregation. Enter the aggregation name and select the levels for the aggregation. Select **`Apply`** to save and validate the aggregation. If the aggregation is a duplicate or it is not in hierarchy order, you will receive an error message.
- Delete an aggregation - select an aggregation from the list box, and then select **`Delete`**.

- Modify an aggregation - select an aggregation from the list box, and then select **Modify**. The bottom panel populates with the aggregation values, and you can select or deselect the levels. When you finish modifying the aggregation levels, select **Apply** to save the changes. Select **Apply** to validate the aggregation. If the aggregation is not in hierarchy order, you will receive an error message.

Here are some guidelines for using Manual Tuning:

- Manual Tuning is available only for MOLAP cubes.
- Levels that are specified for an aggregation must follow at least one existing hierarchy.
- When you modify aggregations, you cannot modify the aggregation name.
- If the cube has an NWAY aggregation, it will display, but it cannot be modified or deleted.
- You cannot add duplicate aggregations in the Manual Tuning window. A duplicate aggregation has the same name as another aggregation or the same list of levels as another aggregation. When you add an aggregation to a cube and select **Apply**, validation will occur to ensure that the aggregation to add is not a duplicate and that its levels follow a hierarchy order.

Using PROC OLAP to Tune Aggregates

To modify an aggregation through PROC OLAP, use the DROP_AGGREGATION statement to delete the aggregation, and then use the AGGREGATION statement to define the new aggregation.

- DROP_AGGREGATION level-name1 < level-name2 ...level-nameN > / NAME=aggregation-name;
- AGGREGATION level-name1 < / TABLE=libname.dataset ></ NAME=' aggregation-name'>;

For more information about the DROP_AGGREGATION and AGGREGATION statements, see “The OLAP Procedure” on page 92.

Multiple Language Support and Dimension Table Translations

OLAP cube data is often generated in one language but needed in other languages. For example, a company's OLAP cube data might be stored in English, but users who speak Spanish and Turkish need access to it. So, the member values as well as the captions that are assigned to dimensions, hierarchies, levels, etc., need to be translated. Multiple language support is available only for cubes that are loaded from star schemas. It is used to read your alternate locale data sets and create locale-specific metadata for use at query time. Query results are returned in the language of the requested locale.

You can specify language support when building a cube either in the Cube Designer Wizard or with PROC OLAP code. There are 56 possible language locales, and English is the default language.

SAS OLAP Cube Studio and Dimension Table Translations

In the Cube Designer - General window, select the **Advanced** button. If you selected Star Schema as the input type in the Cube Designer - General window, you will see the Dimension Table Translations tab. From the Available Languages/Locales list box,

select the needed languages for the translation tables. The first language in the Selected Languages/Locales list box is the default language.

PROC OLAP and the USER_DEFINED_TRANSLATIONS Statement

The USER_DEFINED_TRANSLATIONS statement is used in conjunction with the DIMENSION statement options DIMTABLEMEMPREF= and DIMTABLELIBREF=. For more information, see the "DIMENSION Statement" on page 99.

SAS Servers and Character Encoding

If your server metadata contains characters other than those typically found in English, then you must be careful to start your server with an encoding= or locale= system option that accommodates those characters. For example, a SAS server started with the default US English locale cannot read metadata that contains Japanese characters. SAS will fail to start and log a message indicating a transcoding failure.

In general, different SAS jobs or servers can run different encodings (such as ASCII/EBCDIC or various Asian DBCS encodings) as long as the encoding that is used by the particular job or server can represent all the characters of the data being processed. In the context of server start up, this requires that you review the characters used in the metadata describing your server (as indicated by the server= objectserverparm) to ensure that SAS runs under an encoding that supports those characters.

Adding SAS System Options to a Cube

When you build an OLAP cube, it is often necessary to include additional SAS code that must run prior to the creation of the cube. This includes the creation of user-written formats, PROC statements, and format search paths for the formats that are used on input tables. The Advanced Cube Options window that is accessed from the Cube Designer - General window provides the two entry tabs, Submit SAS Code and Format Search Path. Both tabs provide entry fields for SAS code. You can enter any text in the fields. There is no validation of the text that is entered. However, error messages are sent to the SAS log.

The text is saved to the cube metadata in the SAS Metadata Repository and is used every time the cube is created. You can edit or remove the text after it is initially entered. Highlight the text and make any needed changes, or use the Delete key to remove the text. Select **OK** to save your changes.

Submit SAS Code
: You can use this field to enter a PROC statement or any SAS code that you want to submit before the cube is created. SAS code is submitted before any format search path.

Format Search Path
: You can use this field to enter names of catalogs or libraries for the format search path. The catalogs and libraries must be separated by a blank and will be searched in the order in which they are listed. You use the SAS system option FMTSEARCH= here.

  Note: For more information about SAS formats, see "Formats" in *SAS Language Reference: Dictionary*. △

Note: When you build a cube with SAS OLAP Cube Studio, the format search path is saved with the cube metadata in the SAS Metadata Repository and used every time the cube is recreated. However, if you submit PROC OLAP code through a SAS session,

outside of SAS OLAP Cube Studio, the format search path is ignored. PROC OLAP will not read the information from the SAS Metadata Repository or write the information to the SAS Metadata Repository. △

Specifying Tuning and Performance Options in Cube Aggregations

When you build cubes, you can set various options that improve and optimize cube creation and query performance. These options can be set for all aggregations in a cube or for a specific aggregation. Additionally, these options can be set by using the PROC OLAP options or in SAS OLAP Cube Studio. These options are stored with the cube metadata in the SAS Metadata Repository.

Setting Options on the Cube Designer Wizard

In the Cube Designer - Generated Aggregations window, an Advanced button is provided for access to tuning options. Select the **Advanced** button to open the Performance Options window. There are two tabs for setting tuning options, the Global tab and Aggregation tab.

Global Tab

The global performance options are applied to all aggregations for the cube. These performance options include the

- amount of memory (in megabytes) that is available for aggregation creation
- maximum number of threads that are used to create an aggregation index
- number of aggregations to create in parallel
- partition size (in megabytes) of aggregation table partitions
- number of observations (in kilobytes) to include in the index component file segment
- location of index component files
- location of partitions in which to place aggregation table data
- aggregation tables that are stored in compressed format.

For specific information about these functions, see the Performance Options - Global tab in *SAS OLAP Cube Studio Help*.

Aggregation Tab

The aggregation-specific performance options are applied to an individual aggregation for the cube and override the global option settings for that aggregation. You can define and modify performance options for an aggregation or delete options for an aggregation. The aggregation-specific performance options include the

- partition size (in megabytes) of aggregation table partitions
- number of observations (in kilobytes) to include in the index component file
- location of index component files
- location of partitions in which to place aggregation table data
- aggregation tables stored in compressed format
- aggregations created with indexes

For specific information about these functions, see the Performance Options - Define Aggregation Options dialog box in *SAS OLAP Cube Studio Help*.

Setting Options with PROC OLAP

You can set options for all aggregations in a cube or for a specific aggregation. To set options for all aggregations, set the options in the PROC OLAP statement. To set options for a single aggregation, set the options in the PROC OLAP - Aggregation statement. The options include

INDEXSORTSIZE=n
specifies the amount of memory in megabytes that is available when aggregations are created. The default is the system's available memory.

MAXTHREADS=n
specifies the maximum number of threads that are used to asynchronously create the aggregation indexes. The processing engine calculates how many threads are needed based on the number of indexes that are created and the INDEXSORTSIZE= value. This option sets a limit on the number of threads regardless of the number that is calculated by the processing engine. However, if the processing engine determines that fewer than the maximum number of threads is needed, then only the calculated number of threads are used. The default is the value of the SAS system option SPDEMAXTHREADS or 0.

CONCURRENT=n
specifies the number of aggregations to create in parallel. This option does not apply to the NWAY, which is always built first (unless the NO_NWAY option is set). The default is a maximum of two, based on the results of a special algorithm that takes into consideration the number of aggregations that are being created and the number of processors that are available. The algorithm assumes that CPU resources should be saved for creating aggregation indexes.

WORKPATH=path-name
specifies one or more locations for temporary work files. For all platforms except MVS and VMS, if the WORKPATH option is not specified, then PROC OLAP uses the SPDEUTILLOC system option. If SPDEUTILLOC is not specified, then PROC OLAP uses the UTILLOC system option. If UTILLOC is not specified, then PROC OLAP uses the SAS WORK library path. For MVS and VMS, then PROC OLAP uses the SPDEUTILLOC system option. If neither WORKPATH= nor SPDEUTILLOC is specified for MVS and VMS, the following error is returned: "ERROR: No valid work path has been given. Use the WORKPATH= PROC OLAP option or the SPDEUTILLOC= system option to set a valid work path."

DATAPATH=('path-name' ...'pathnameN')
specifies the location of one or more partitions (.DPF files) in which to place aggregation table data. The data is distributed by cycling through each partition location according to the partition size (set by using the PARTSIZE= option). The default is the cube subdirectory of the location that is specified by the PATH= option in the PROC OLAP statement.

INDEXPATH=('path-name' ...'pathnameN')
specifies the locations of the index component files (.IDX and .HYB files) that correspond to each aggregation table partition as specified by the DATAPATH= option. The default is the cube subdirectory of the location that is specified by the PATH= option in the PROC OLAP statement.

COMPRESS | NOCOMPRESS
specifies whether or not to store the aggregation tables in a compressed format on disk. The default is NOCOMPRESS.

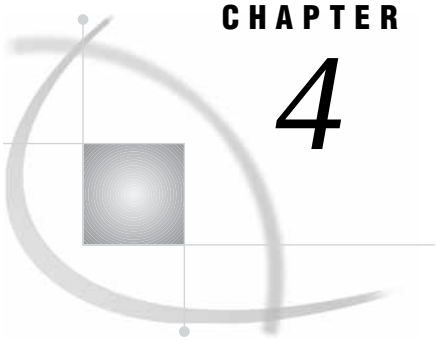
INDEX | NOINDEX
specifies whether or not to create the aggregations with indexes. For faster cube creation and adding and deleting aggregations, you can set this option to NOINDEX. However, the lack of indexes will adversely affect query performance. The default is INDEX.

PARTSIZE=size-in-megabytes
specifies the partition size in megabytes of the aggregation table partitions (.DPF files) and their corresponding index components (.IDX and .HYB files). The default is 128 megabytes. The minimum value is 16 megabytes.

SEGSIZE=number-of-rows-in-kb
specifies the number of observations (table rows) in kilobytes to include in the index component file segment. The minimum size is 1k (1,024 rows), so the value of SEGSIZE= is a multiple of 1024 as expressed in kilobytes. The segmented indexes are used to optimize WHERE-expression processing. Each parallel thread is given a segment of the table to evaluate that is equal to the SEGSIZE= value. The default is 8 kilobytes (8,192 rows). The minimum is 1 kilobyte (1,024 rows).

Note: INDEXSORTSIZE=, MAXTHREADS=, and CONCURRENT= are only available in the PROC OLAP statement. △

For more information about these options, see “PROC OLAP Statement” on page 92 and “AGGREGATION Statement” on page 110.



CHAPTER

4

Using SAS OLAP Cubes

Using a Cube with ADO MD

Applications gain access to SAS OLAP cubes through ADO MD. *ADO MD* is an industry standard programming interface to multidimensional data. It offers the same functionality as OLE DB for OLAP but in a simpler programming model. Accessing SAS OLAP cubes through ADO MD requires the SAS Integrated Object Model (IOM) Data Provider, which is a component of SAS Integration Technologies. The SAS IOM Data Provider is installed with the client-side SAS Integration Technologies 9.0 product. See the *SAS Data Providers: ADO/OLE DB Cookbook* for more information about IOM data provider usage with ADO MD. In particular, see the topics "Reading Multidimensional Data with ADO MD" and "About Schema Rowsets."

Using a Cube with OLE DB for OLAP

In addition to ADO MD, applications gain access to SAS OLAP cubes through *OLE DB for OLAP*, an industry standard set of programmable Component Object Model (COM) interfaces that expose multidimensional data. For SAS OLAP cubes, the OLE DB interfaces are exposed by the SAS IOM Data Provider, a component of SAS Integration Technologies. The SAS IOM Data Provider enables applications to perform data analysis by providing a means to view schema information, submit MDX queries, and retrieve results. The SAS IOM Data Provider is installed with the client-side SAS Integration Technologies 9.0 product. See the *SAS Data Providers: ADO/OLE DB Cookbook* for more information about IOM data provider usage. In particular, see the topics "About OLE DB Interfaces" and "About Schema Rowsets."

Using a Cube with Additional SAS Software

Several additional SAS products gain access to SAS OLAP cubes:

- Enterprise Guide
- SAS AppDev Studio
- SAS Desktop Report Studio
- SAS Information Map Studio
- SAS Web Report Studio
- SQL Pass-Through Facility for OLAP. For more information, see "Accessing OLAP Cubes from SAS: SQL Pass-Through Facility for OLAP" on page 40.

  Note: The SQL Pass-Through Facility for OLAP does not require additional licensing. △

See the product help and documentation for these products for information on how to access SAS OLAP cubes.

Using a Cube with Third-Party Clients

The SAS OLAP Server exposes multidimensional data through OLE DB for OLAP interfaces. Supporting these industry standard interfaces enables the SAS OLAP Server to integrate with third-party clients. The user interfaces for these clients vary widely. To ensure successful integration with the SAS OLAP Server, usage guidelines for some third-party clients have been established.

Microsoft Excel 2000 and Excel 2002 PivotTable

To view SAS OLAP cubes in Microsoft Excel, you must identify the server where your cubes are stored and the cube you want to analyze with the *PivotTable and PivotChart* Wizard.

1. In Microsoft Excel, select

   Data ▶ PivotTable and PivotChart Report

   This opens the PivotTable and PivotChart Wizard.
2. At the PivotTable and PivotChart Wizard, select the radio buttons for **External data source** and **PivotTable**. Select **Next**.
3. At the PivotTable and PivotChart Wizard, select **Get Data**. This opens the Choose Data Source window.
   - On the OLAP Cubes tab, select **<New Data Source>**.
   - Select **OK**. This opens the Create New Data Source window.
     - In field 1, enter the name that you want to associate with the cube data you are accessing.

       Note: This is the name that Excel uses to store your work. △
     - In field 2, select **SAS OLAP Data Provider 9.1**.
     - In field 3, select the **Connect** button. This opens the Data Link Properties window. In the Data Source field, enter the name of the SAS 9 OLAP Server you are accessing. If necessary, enter your user ID and

password for accessing the server. Select the **SAS Protocol** (Bridge, Com, or Corba). If you select **Bridge**, then you must specify the SAS Service Name/Port. Select **OK**.

- If the connection to the server is successful, field 4 will be active.

  Note: A successful connection depends on several factors such as accurate data source and port information, accurate user account information, and whether the server is running and can be accessed. For details about SAS Protocol, SAS Service Name, and other connection properties, see "SAS OLAP Provider Connection Properties" in the *SAS Data Providers: ADO/OLE DB Cookbook*. △

  Select the cube that you want to analyze in Excel.

  Note: If field 4 is active, but there are no cubes shown in the drop down box, this means the OLAP server was unable to locate any cubes. A possible cause is that your OLAP server is not associated with the correct OLAP schema. You can determine the OLAP schema assigned to your OLAP server from the SAS Management Console. Then in SAS OLAP Cube Studio, verify that the OLAP schema has cubes. For more information about OLAP schemas see the *SAS Intelligence Architecture: Planning and Administration Guide*. △

- You must select the radio button **Save my userID and password in the data source definition** when you connect to a secure server. If you select the radio button, then a message window opens that informs you how the user ID and password will be stored in the data source definition. At this point you must confirm your selection of the radio button. There are variations in performance between Microsoft Excel 2000 and Microsoft Excel 2002. If you do not select the radio button, then when you connect to a secure server, the connection will fail in one of these ways:
  - In Microsoft Excel 2000, you will return to the Choose Data Source window in the PivotTable and PivotChart Wizard.
  - In Microsoft Excel 2002, the DataLink Properties window reloads, which allows you to re-enter the data source information.
- When you finish entering information in the Create New Data Source window, select **OK**. At this point Excel saves the data as a Microsoft query (OQY) file that you can later reference and load in Excel.

After completing the fields in the Create New Data Source window, you return to the Choose Data Source window. You see your new data source listed on the OLAP Cubes panel. Select **OK**. This returns you to the PivotTable and PivotChart Wizard window. Select **Next**. This loads the PivotTable and PivotChart Wizard.

4. At On the PivotTable and PivotChart Wizard, select where you want to put the PivotTable. You can choose to place the PivotTable in a New worksheet or in the Existing worksheet. Select **Finish**. The PivotTable is loaded.
5. At the worksheet and PivotTable menu, select cube data items to populate the columns and rows of the PivotTable. Use the drag-and-drop selection method to move the data buttons to the PivotTable. For the PivotTable data area, select the data buttons that are measures of the cube data.
6. From this point, use standard Microsoft Excel functionality to view cube data.

Note: You must have Microsoft Query installed to view OLAP cubes in Microsoft Excel. △

Saving a PivotTable as a Web Page

When you save a PivotTable as a static Web page in Microsoft Excel 2002, the user password is not automatically saved with the HTML file that is generated. For Excel 2002 PivotTables that are connected to a secure OLAP server, this can result in an error message when the HTML file is loaded into Internet Explorer.

In Microsoft Excel 2002, when you select

File ► Save as Web Page

a message window displays stating that the user password will not be saved. You then have an opportunity to continue saving the HTML file or not. If you choose to save the HTML file without the password, an error message might be displayed when the user tries to load the HTML file in Microsoft Internet Explorer. To prevent this error message, you can manually edit the HTML file to include the user password. For example, in this connection tag, you would add the password option, **password=mypass;** to the <Connection> string.

```
<Connection>Provider=sas.OLAPProvider.9.1;User ID=&userid;Data
Source=SAS OLAP Provider Server;Location=&location;
Mode=ReadWrite|Share Deny None;SAS Logical Name="";
SAS Machine DNS Name=&SASMachineDNSName ;
SAS Port=&Port;SAS Protocol=2;
SAS Server Type=2</x:Connection>
```

Microsoft Office Web Components 2000 and 2002 PivotTable

For the Microsoft Office Web Components 2000 and 2002 PivotTable user interface, the PivotTable connection string must specify the provider and data source. Specifically, the connection string must specify

- Provider=sas.IOMProvider.9.1
- the data source as a URL/URI style string that contains all the necessary connection properties:

  ```
  olapServerName&Property1=value&property2=Value&property3=value
  ```

  The string must begin with the name of the OLAP Server to which you are connecting. To list additional connection properties, follow the server name with "&" and list "property name=property value" pairs, delimited by "&".

  Note: All connection properties, with the exception of provider, must be specified within the data source string. △

  Note: For a description of these properties see the *SAS Data Providers: ADO/OLE DB Cookbook*. In particular, see the sections "Connections and Data Sources" and "Connecting to a Remote SAS OLAP server with the IOM Provider." △

Here is an example of an HTML page using a Microsoft Office 2000 PivotTable in conjunction with the SAS OLAP Server. The data source string specifies an OLAP server with the name mktg.unx.com, a SAS port of 6176, and the ProtocolBridge for SAS protocol defined as 2.

```
<HTML>
<BODY ONLOAD = "Setup_PT()">
<OBJECT id="PivotTable1"
      CLASSID="CLSID:0002E520-0000-0000-C000-000000000046"
      style="HEIGHT: 500px; WIDTH: 500px">
</OBJECT>
```

```
<SCRIPT language=VBScript>
   Sub Setup_PT()

constr=''Provider=sas.iomprovider.9.1;Data Source=mktg.unx.com&SAS Port=6176
&Location=localhost&SAS Protocol=2''
      PivotTable1.ConnectionString=constr
      PivotTable1.DataMember="CAMPAIGN"
      PivotTable1.ActiveView.AutoLayout

   End Sub
</SCRIPT>

</BODY>
</HTML>
```

For Microsoft Office 2002 PivotTable Web Components, use CLASSID="CLSID:0002E552-0000-0000-C000-000000000046".

ProClarity Professional

You can access SAS OLAP cubes from within ProClarity Professional. The following steps show you how to load a SAS OLAP cube for analysis:

1. From the File menu, select

   Options ▶ OLAP Provider ▶ Change Provider

   SAS OLAP Data Provider 9.1 ▶ OK

   On the OLAP Provider tab of the Options window, SAS OLAP Data Provider 9.1 should now appear as the current provider. Select **OK**.
2. From the File menu, select **Open Cube**. This opens the Connect window.
3. In the **Server** field, enter the name of the SAS OLAP server that you are connecting to as well as the user name and password. Select **OK**. This loads the Open Cube window.
4. In the Open Cube window, select a cube to open. Select **OK**. The cube is loaded into the ProClarity workspace.
5. From this point, use standard ProClarity functionality to manipulate the SAS OLAP cube.

Note: The ProClarity interface currently enables you to specify the data source that you are connecting to as well as the user name and password. However, it does not allow you to specify other provider properties that might be necessary to establish a server connection such as SAS Port or SAS Protocol. For further information about how to set additional SAS Provider properties, see the *SAS Data Providers: ADO/OLE DB Cookbook*. △

Note: In ProClarity you can access and view measures by selecting

View ▶ MDX Editor

The cube measures are displayed in the lower metadata window. However, when viewing SAS OLAP cubes in ProClarity, user-defined measures are not recognized. △

CHAPTER 5

Transitioning from SAS OLAP Server Release 8.2 to SAS 9.1

Conversion and Migration Issues from Release 8.2 to SAS 9.1

Data Conversion and Migration

Version 8 OLAP data is accessible in SAS 9.1. However, when you build or read Version 8 MDDBs, you will not be able to take advantage of the new optimization features of SAS 9.1 such as the threaded kernel, without re-creating the data as SAS 9.1 data. SAS OLAP data structures have changed from Version 8 to SAS 9.1 in the following ways:

Data Sets
: In Version 8, any library reference that was assigned to a folder that contained a Version 6 data set was automatically assigned using the Version 6 engine, and new data sets that were written to that library were Version 6 data sets. In SAS 9.1, a libref is assigned using the SAS 9.1 engine regardless of whether the data sets in that library were generated in Version 8 or SAS 9.1. All new data sets are written as SAS 9.1 data.

MDDBs
: Between Version 8 and SAS 9.1, the file structure of a cube changed. A Version 8 MDDB is a single physical file in the file system (the file extension on both Windows and UNIX platforms is .sas7bmdb). In the folder, there are files that represent the cube, its dimensions, levels, members, and aggregations. Therefore, the only way to convert a Version 8 MDDB to a SAS 9.1 cube is to rebuild it from the input data by using PROC OLAP.

Code Conversion and Migration

PROC MDDB versus PROC OLAP

There is no automated method to convert Version 8 MDDB code to SAS 9.1 OLAP code. PROC MDDB executes in SAS 9.1. However, it creates a Version 8 MDDB that cannot use the SAS 9.1 features such as the threaded kernel and the improved caching mechanism. To help you rewrite your MDDB code in PROC OLAP syntax, see "Comparing PROC MDDB Code and PROC OLAP Code" on page 89.

Data Model and Viewer Customizations

A number of Version 8 overrides and customizations might not be needed when you rework an application in SAS 9.1, because a number of the new features in SAS 9.1 resolve these Version 8 issues. Some of these new features are documented in What's New in the OLAP Server in SAS 9.1.

All applications that were written in earlier versions of SAS will work using the legacy technology. However, these older systems cannot use any of the new features in SAS 9.1 or the new OLAP or SAS metadata servers, and vice versa.

Repository Conversion and Migration

You can use your current metadata in the SAS Metadata Repository and the metabases repositories within legacy environments. However, as with the data and applications, legacy queries (Version 8 and earlier) will run using the Version 8 engine and not be able to use the enhanced features in SAS 9.1. If changes are made to a Version 8 repository by using SAS 9.1 (such as adding tables to the repository), those changes will work properly when accessing that repository in Version 8. Also important are

- standard CMR to SAS Metadata Repository migration
- customizations, add-ons, and attribute dictionaries.

For more information, see "Comparing OLAP Functionality in SAS 8 and SAS 9.1" on page 86 and "Comparing PROC MDDB Code and PROC OLAP Code" on page 89.

Comparing OLAP Functionality in SAS 8 and SAS 9.1

The following table lists some of the basic tasks that are performed when you create a SAS OLAP cube and the differences in how these tasks are handled in SAS Version 8 and SAS 9.1.

Functionality	How it is handled in Version 8	How it is handled in SAS 9.1
Defining hierarchy levels for dimensions	PROC MDDB CLASS statement	PROC OLAP LEVEL statement
Specifying statistics	PROC MDDB VAR statement In PROC MDDB, you specify stored statistics. Derived statistics are available at query time.	PROC OLAP MEASURE statement In PROC OLAP, you have one MEASURE statement for each combination of an input column plus its associated statistic (which includes defining derived statistics).
Grouping levels for drill-down capability	PROC MDDB HIERARCHY statement option, DISPLAY=YES/NODATA option In PROC MDDB, individual classes (levels) are not linked but can be grouped into drill-down hierarchies by using the DISPLAY option in the HIERARCHY statement .	PROC OLAP DIMENSION statement PROC OLAP uses the DIMENSION statement to accomplish the same goal. Each dimension consists of a hierarchy that is composed of grouped levels. Each level is fed by one input column.

Functionality	How it is handled in Version 8	How it is handled in SAS 9.1
Creating NWAYs and other aggregations	Implicitly, PROC MDDB always creates an NWAY subtable, which is the combination of all class (level) variables. Any additional subtables can be explicitly requested by using HIERARCHY statements.	In PROC OLAP, subtables are called aggregations. As with PROC MDDB, the NWAY aggregation is created by default; although you can choose not to create the NWAY. Aggregation creation is handled by the AGGREGATION statement.
Specifying hierarchy navigation	HIERARCHY statement, DISPLAY=YES/NODATA option In PROC MDDB, the HIERARCHY statement can be used to specify navigation hierarchy and, unless you specify DISPLAY=NODATA, also to create an aggregation.	HIERARCHY statement In PROC OLAP, the navigation hierarchies are handled by the HIERARCHY statement.
Loading cubes from star schemas	Star schema support is available through the Distributed Multidimensional Metabase (DMM) facility.	You can load your cubes from star schemas and indicate your dimension tables by using the DIMTABL=, DIMKEY=, and FACTKEY= options in the DIMENSION statements. You can also use a specified star schema instead of an NWAY aggregation by using the NO_NWAY option in the PROC OLAP statement. To load the cube from a star schema, use the FACT= option instead of the DATA= option.

Functionality	How it is handled in Version 8	How it is handled in SAS 9.1
Using externally summarized data sources	HOLAP data groups You use HOLAP data groups to point to externally summarized data sources, which might be available through RDBMS tables, SAS data sets, or, in the case of multiple aggregations combined in one table, a PROC SUMMARY output data set with multiple types.	AGGREGATION statement or TABLE= option You use PROC OLAP's AGGREGATION statement to point to externally summarized sources
Cube metadata registration	The SAS/EIS metabase facility and the Distributed Multidimensional Metadata (DMM) facility for HOLAP data groups serve as an extension to the MDDB's own structural information. Information about MDDBs, such as drill-down hierarchies and distributed data sources, is stored in those repositories and accessed there by the different data model extensions.	All relevant structural information is contained within the cube and most of it is also replicated within the SAS Open Metadata Architecture. This is done to gain the following advantages: □ the ability to disassociate the cube definition process from cube creation. Later, you can create a cube using its stored definition □ the ability to define and enforce security at the SAS Open Metadata Architecture level □ the ability to manage and control the data source in the centralized SAS Open Metadata Architecture.

Comparing PROC MDDB Code and PROC OLAP Code

The following example code illustrates the difference between PROC MDDB and PROC OLAP when you create a simple cube.

PROC MDDB Sample Code

To create a Version 8 MDDB and make it ready for use with SAS or external viewers, you run PROC MDDB. After you run the code, you register the cube in the SAS/EIS metabase facility in order to transform the display hierarchies into the HIERARCHY attribute of the metabase registration. For partitioned or distributed cubes, you must

also create a HOLAP data group definition by using the Distributed Multidimensional Metadata (DMM) facility.

```
proc mddb data=sashelp.prdsale out=work.prdmddb;
    class country region division prodtype product year quarter month;
    var actual / sum n;
    var predict / sum n;
    hierarchy year quarter month / name='Time YQM' display=nodata;
    hierarchy year month / name='Time YM' display=nodata;
    hierarchy country region division / name='Geography' display=nodata;
    hierarchy prodtype product / name='Product Line' display=nodata;
    hierarchy country prodtype year;
run;
```

PROC OLAP Sample Code

To create the same cube in SAS OLAP Server 9.1, you use the following PROC OLAP code:

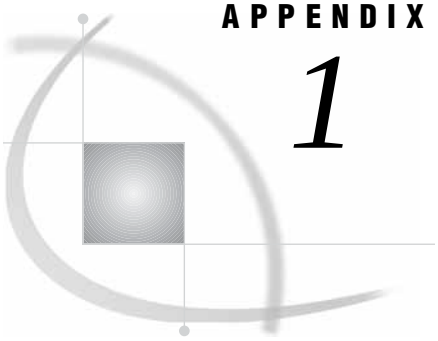
Note: You can also create this cube by using the Cube Designer wizard, which is available in SAS OLAP Cube Studio and SAS ETL Studio. △

```
proc olap data=sashelp.prdsale cube=prdcube path='c:\tmp';
    metasvr olap_schema='banking schema'
            repository='financial repository'
            host='misdept.us.mar.com'
            port=9999
            protocol=com
            userid=jjones
            pw='my password';
     dimension Geography hierarchies=(Geography);
       hierarchy Geography levels=(country region division);
       level country caption='Country';
       level region caption='Region';
       level division caption='Division';

    dimension Time hierarchies=(Time);
       hierarchy Time caption='Time' levels=(year quarter month);
       hierarchy Time caption='Time' levels=(year month);
       level year caption='Year' type=year ;
       level quarter caption='Quarter' type=quarters ;
       level month caption='Month' type = months;

    dimension prdln caption='Product Line' hierarchies=(prdln);
       hierarchy prdln caption='Product Line' levels=(prodtype product);
       level prodtype caption='Product Type';
       level product caption='Product Name';

    measure actual_sum caption='Actual Sum' stat=sum column=actual;
    measure actual_n caption='Actual Count' stat=n column=actual;
    measure predict_sum caption='Predict Sum' stat=sum column=predict;
    measure predict_n caption='Predict Count' stat=n column=predict;
    aggregation country prodtype year /name='Product Types by Country';
run;
```



APPENDIX

1

The OLAP Procedure

The OLAP Procedure

The OLAP procedure is one of the tools in SAS 9.1 that you can use to create and delete cubes, and add and delete aggregations.

Note: You can also use the Cube Designer wizard to maintain OLAP cubes. The Cube Designer wizard can be launched from SAS ETL Studio and SAS OLAP Cube Studio. Help on using the wizard to build cubes is available from within both applications. △

In addition to the basic cube creation tasks, PROC OLAP also enables you to

- build cubes with ragged hierarchies
- control options that can be used to optimize cube creation and query performance
- specify data set options on detail, fact, dimension, and drill-through tables
- create TIME dimensions
- design dimensions that have more than one hierarchy
- define global calculated members and named sets
- include SAS code when you submit PROC OLAP in batch mode
- read alternate locale data sets and create locale-specific metadata for use at query time

Syntax: OLAP Procedure

PROC OLAP < *option(s)*>;
METASVR OLAP_SCHEMA='*schema-name*' < *option(s)*>;
DIMENSION *dim-name* HIERARCHIES=(*hier–name*) <*option(s)*>;
HIERARCHY *hier-name* LEVELS=(*level–name1*<*level–name2 ... level-nameN*>) <*option(s)*>;
LEVEL*level-name* <*option(s)*>;
PROPERTY *prop-name* LEVEL=*level-name*<*option(s)*>;
MEASURE *measure-name* STAT=*statname* < *option(s)*>;
AGGREGATION *level-list* </*options*>;
DROP_AGGREGATION *level-name1* < *level-name2 ... level-nameN* > / NAME='*aggregation-name*' ;
DEFINE MEMBER | SET '*member-or-set-name*' AS '*mdx-expression*' ;
USER_DEFINED_TRANSLATIONS one or more of 56 locale specifications;

PROC OLAP Statement

The PROC OLAP statement specifies the input data source, cube name, and path. This statement can also be used to

- specify options that might improve query performance
- delete cubes
- specify global settings for handling missing hierarchy members in ragged hierarchies.

PROC OLAP <*option(s)*>;

Options

Note: For information about options that can be used to optimize cube creation and query performance, see Syntax Options Used for Performance"Syntax Options Used for Performance" on page 128. △

DATA | FACT=*dsname*
specifies the data source for the cube. The unsummarized data source can be any SAS data file, including files that are supported by SAS access engines. If you load the cube from a star schema, then the *dsname* is the name of the fact table that contains the analysis variables from which to derive the measures for the cube. The fact table must also contain fact keys that correspond to dimension tables in the star schema.

You can also provide data set options along with DATA | FACT=. Options are stored within the cube and reapplied when the data is accessed at run time. For more information, see "Data Set Options" in *SAS Language Reference: Concepts*.

Note: This option is not required if you want to define the cube by using input data from a fully summarized external data source (a crossing of all dimensions known as an NWAY); in that case, you specify the data source for the cube by using the TABLE= option in the AGGREGATION statement. △

Interaction: If you load the cube from a star schema, then you must use the DIMENSION statement to

- specify the dimension table name (the DIMTBL= option)
- specify the dimension (primary) key column (the DIMKEY= option)
- specify the column (foreign key) in the fact table that corresponds to the dimension key column (the FACTKEY= option).

DRILLTHROUGH_TABLE | DT_TABLE | DT_TBL=
specifies an optional drill-through table. Drill-through tables can be used by client applications to provide a view from processed data into the underlying data source. You can specify the DATA | FACT= table or a different table that includes the necessary data and columns.

You can also specify data set options with this option. Options are stored within the cube and reapplied when the data is accessed at run time. For more information, see "Data Set Options" in *SAS Language Reference: Concepts*.

CUBE=*cube-name*
specifies a valid SAS name for the cube to be created or updated. For naming guidelines, see Naming Guidelines for SAS OLAP Server"Naming Guidelines for SAS OLAP Server" on page 120.

PATH='*path-name*'
specifies the physical or logical path to the location of a new cube. Within the specified path, the cube is stored in a directory that uses the name of the cube in

upper-case letters. For example, if you enter the path **'c:\v9cubes'** and the cube name is **MrktData**, then the cube is stored in **'c:\v9cubes\MKRTDATA'**. Enclose the path within quotation marks.

DESC | DESCRIPTION='*cube-description*'
specifies any number of characters to be stored as descriptive text. If the text includes blank spaces or any characters that are not permitted in a valid SAS name, then enclose the text within quotation marks.

NO_NWAY
prevents PROC OLAP from automatically creating an NWAY aggregation (the crossing of all dimension levels) for the new cube. The automatically created NWAY is usually the largest in the cube and most resembles the content of the unsummarized data source.

Interaction: If you use this option, then the input data source that is specified with the DATA= or FACT= option must be available at run time; otherwise, queries that are not covered by other aggregations will fail.

EMPTY_CHAR='*string*'
for ragged hierarchies, this option specifies the string that identifies missing characters as hierarchy members. For example, by default, SAS identifies missing characters as a blank space. To identify blank spaces as missing hierarchy members, enter **''**as the value of the EMPTY_CHAR= option. To identify every instance of the word "empty" as missing hierarchy members, enter **'empty'**. In this case, hierarchy members that have the value **'empty'** are treated as missing for all levels in which the hierarchy appears, and they are ignored when you drill down into the cube. If you enclose the string within quotation marks, then you can use a maximum of 256 characters. If you do not use quotation marks, then the string must be a valid SAS name. For more information, see Naming Guidelines for SAS OLAP Server"Naming Guidelines for SAS OLAP Server" on page 120.

Interaction: You can use the HIERARCHY statement to override this string for specific hierarchies.

EMPTY_NUM='*string*'
for ragged hierarchies, this option specifies the string that identifies missing numeric values as hierarchy members. For example, by default, SAS identifies missing numeric values as a single period (.). To identify periods as missing hierarchy members, enter **'.'** as the value of the EMPTY_NUM= option. To identify every instance of the value "123" as missing hierarchy members, enter **'123'**. In this case, hierarchy members that have the value **'123'** are treated as missing for all levels in which the hierarchy appears, and they are ignored when you drill down into the cube. If you enclose the string within quotation marks, then you can use a maximum of 256 characters. If you do not use quotation marks, then the string must be a valid SAS name. For more information, see Naming Guidelines for SAS OLAP Server"Naming Guidelines for SAS OLAP Server" on page 120.

Note: If there is no format associated with the member value, then **BEST12** is used as the format. △

Interaction: You can use a HIERARCHY or LEVEL statement to override this string of specific hierarchies.

INDEXSORTSIZE=*n*
specifies the amount of memory in megabytes that is available when aggregations are being created.

Default: The system's available memory

Tip: So that each built index has a fair share of the assigned INDEXSORTSIZE memory, INDEXSORTSIZE is divided by the CONCURRENT value. The value of INDEXSORTSIZE should give each concurrent index build enough memory to at least hold a table PARTSIZE. For best performance, INDEXSORTSIZE divided by CONCURRENT should be greater than PARTSIZE.

MAXTHREADS=*n*
specifies the maximum number of threads that are used to asynchronously create the aggregation indexes. The processing engine calculates how many threads are needed based on the number of indexes that are being created and the INDEXSORTSIZE= value. This option sets a limit on the number of threads regardless of the number that is calculated by the processing engine. However, if the processing engine determines that fewer than the maximum number of threads is needed, then only the calculated number of threads are used.

Default: The value of the SAS system option SPDEMAXTHREADS or 0. If the value is 0, then the processing engine determines the number of threads based on the number of indexes that are created plus the available memory. The maximum value is 65,536 threads.

CONCURRENT=*n*
specifies the number of aggregations to create in parallel. This option does not apply to the NWAY, which is always built first (unless the NO_NWAY option is set).

Default: A maximum of two, based on the results of a special algorithm that takes into consideration the number of aggregations that are being created and the number of processors that are available. The algorithm assumes that CPU resources should be saved for creating aggregation indexes.

Tip: So that each built index has a fair share of the assigned INDEXSORTSIZE memory, INDEXSORTSIZE is divided by the CONCURRENT value. The value of INDEXSORTSIZE should give each concurrent index build enough memory to at least hold a table PARTSIZE. For best performance, INDEXSORTSIZE divided by CONCURRENT should be greater than PARTSIZE.

WORKPATH=('*path-name*' ...'*pathnameN*')
specifies one or more locations for temporary work files.

Default: For all platforms except MVS and VMS, if the WORKPATH option is not specified, PROC OLAP uses the SPDEUTILLOC system option. If SPDEUTILLOC is not specified, PROC OLAP uses the UTILLOC system option. If UTILLOC is not specified, PROC OLAP uses the SAS WORK library path. For MVS and VMS, PROC OLAP uses the SPDEUTILLOC system option. If neither WORKPATH= nor SPDEUTILLOC is specified for MVS and VMS, the following error is returned: **`ERROR: No valid work path has been given. Use the WORKPATH= PROC OLAP option, or the SPDEUTILLOC= system option to set a valid work path.`**

DATAPATH=('*path-name*' ...'*pathnameN*')
specifies the location of one or more partitions (.DPF files) in which to place aggregation table data. The data is distributed by cycling through each partition location according to the partition size (set using the PARTSIZE= option). For example, if you specify **`DATAPATH=('c:\data1' 'd:\data2')`**, then PROC OLAP places the first partition of each aggregation table into directory c:\data1, the second partition of each table into directory d:\data2, the third partition of each table into c:\data1, and so on. It is also possible to have aggregation tables that use less than the specified number of partitions. For example, your cube might contain an aggregation table that fits entirely into c:\data1.

Note: This option applies to the automatically created NWAY and all aggregations that do not explicitly specify a DATAPATH= option in the AGGREGATION statement. △

Default: The cube subdirectory of the location that is specified by the PATH= option in the PROC OLAP statement

INDEXPATH=('*path-name*' ...'*pathnameN*')
specifies the locations of the index component files (.IDX and .HYB files) that correspond to each aggregation table partition as specified by the DATAPATH= option.

Note: This option applies to the automatically created NWAY and all aggregations that do not explicitly specify an INDEXPATH= option in the AGGREGATION statement. △

Note: Indexes are not created for aggregations that have fewer than 1,024 records. △

Default: The cube subdirectory of the location that is specified by the PATH= option in the PROC OLAP statement

COMPRESS | NOCOMPRESS
specifies whether or not to store the aggregation tables in a compressed format on disk.

Note: This option applies to the automatically created NWAY and all aggregations that do not explicitly specify a COMPRESS option in the AGGREGATION statement. △

Default: NOCOMPRESS

INDEX | NOINDEX
specifies whether or not to create the aggregations with indexes. For faster cube creation and adding and deleting aggregations, you can set this option to NOINDEX; however, the lack of indexes will adversely affect query performance.

Note: This option applies to the automatically created NWAY and all aggregations that do not explicitly specify an INDEX option in the AGGREGATION statement. △

Default: INDEX

PARTSIZE=*size-in-megabytes*
specifies the partition size in megabytes of the aggregation table partitions (.DPF files) and their corresponding index components (.IDX and .HYB files).

Note: This option applies to the automatically created NWAY and all aggregations that do not explicitly specify a PARTSIZE= option in the AGGREGATION statement. △

Default: 128 megabytes. The minimum value is 16 megabytes.

SEGSIZE=*number-of-rows-in-kb*
specifies the number of observations (table rows) in kilobytes to include in the file segment of the index component. The minimum size is 1k (1,024 rows), so the value of SEGSIZE= is a multiple of 1024 as expressed in kilobytes. The segmented indexes are used to optimize WHERE-expression processing. Each parallel thread is given a segment of the table to evaluate that is equal to the SEGSIZE= value.

Note: This option applies to the automatically created NWAY and all aggregations that do not explicitly specify a SEGSIZE= option in the AGGREGATION statement. △

Default: 8 kilobytes (8,192 rows). The minimum is 1 kilobyte (1,024 rows).

DELETE
deletes the physical cube that is specified with the CUBE= option. It also deletes the cube's definition, which is stored in the metadata server. The DELETE option does not delete the input data source.

If either the physical cube or its registration, or both are not present, then the DELETE option behaves as explained in the following table:

Table A1.1 How the DELETE Option Behaves If the Physical Cube or Its Registration Is Not Present

Physical cube exists	Registration exists	DELETE option behaves this way
No	Yes	The physical cube is not deleted. The registration is deleted, not updated. If there is a registration, and you use the DELETE option, the registration is always deleted and you cannot recreate the cube from the registration. You can only recreate the cube from the registration when you use the DELETE_PHYSICAL option.
No	No	Fails because there is nothing to delete.
Yes	No	Fails because the cube cannot be located without its registration information. You must manually delete the cube.

DELETE_PHYSICAL
deletes the physical cube that is specified with the CUBE= option but leaves the cube definition intact. This enables you to build a new cube based on the saved cube definition.

If either the physical cube or its registration, or both are not present, then the DELETE_PHYSICAL option behaves as explained in the following table:

Table A1.2 How the DELETE_PHYSICAL Option Behaves If the Physical Cube or Its registration Is Not Present

Physical cube exists	Registration exists	DELETE_PHYSICAL option behaves this way
No	Yes	Fails because there is no physical cube to delete.
No	No	Fails because there is nothing to delete.
Yes	No	Fails because the cube cannot be located without its registration information. You must manually delete the cube.

METASVR Statement

The METASVR statement identifies the SAS Metadata Repository in which existing cube metadata information exists or in which metadata about a new cube is stored.

METASVR OLAP_SCHEMA='*schema-name*' < *option(s)*>;

The METASVR statement options can be used to override the metadata repository connection values that are specified through SAS start-up options.

Note: During an interactive SAS session, if connection information is not available either through start-up settings or through a METASVR statement, then the user is prompted for the missing information. For more information about SAS start-up options, see *SAS Language Reference: Dictionary*. △

Following is an example of a METASVR statement with all of its options set:

```
METASVR olap_schema='Banking Schema'
            repository='financial repository'
            host='misdept.us.mar.com'
            port=9999
            protocol=com
            userid=jjones
            pw='my password';
```

Required Argument

OLAP_SCHEMA='*schema-name*'
is a string that specifies the name of the schema that has been defined in a SAS Metadata Repository. The name can be a maximum of 32 characters. The OLAP schema specifies which group of cubes that an OLAP server can access. Each OLAP schema can be accessed by multiple OLAP servers; however, each OLAP server has access to only one OLAP schema. When using embedded blanks or special characters in the schema name, enclose the name in quotation marks.

Options

REPOSITORY='*repos-name*'
is a string that specifies the name of the SAS Metadata Repository in which existing cube metadata information exists or in which metadata about a new cube is stored. The name can be a maximum of 60 characters. When using lowercase letters, embedded blanks, or special characters in the repository name, enclose the name in quotation marks.

HOST='*metadata-server-host-name*'
is a string that specifies the IP address of the metadata repository host. An example is **misdept.us.mar.com**. The address can be a maximum of 256 characters. When using lowercase letters, embedded blanks, or special characters in the host name, enclose the name in quotation marks.

PORT=*port-number*
specifies the numeric value of the port on which the metadata repository resides.

PROTOCOL=BRIDGE | COM
specifies the protocol that is used to connect to the specified metadata repository.

USERID='*userid*'
is a string that specifies the user's identification for the specified metadata repository. The identification can be a maximum of 256 characters. When using lowercase letters, embedded blanks, or special characters in the user ID, enclose the user ID in quotation marks.

PW='*password*'
is a string that specifies the password for the user identified with the USERID= option. The password can be a maximum of 512 characters. When using lowercase

letters, embedded blanks, or special characters in the password, enclose the password in quotation marks.

DIMENSION Statement

The DIMENSION statement defines the logical and hierarchical relationships between the variables in the input data.

DIMENSION *dim-name* HIERARCHIES=(*hier–nam ... hier-nameN*) <*option(s)*>;

At least one DIMENSION statement must be specified when the cube is created. The DIMENSION statement is not used when adding or deleting aggregations from cubes. You can have a maximum of 128 dimensions per cube.

Note: The DIMENSION statement does *not* create aggregations. To create aggregations, use the AGGREGATION statement. △

A DIMENSION statement must include the name of at least one hierarchy in its HIERARCHIES= option. In addition, a HIERARCHY statement must include the name of at least one level in its LEVELS= option. However, you cannot use the same level in more than one dimension.

Note: You can use LEVEL statements to specify a time period for each level in a TIME dimension. LEVEL statements are also used to supply information such as a level-specific sort order or a level description. △

The following example uses one DIMENSION statement, two HIERARCHY statements, and three optional LEVEL statements to define a fully specified dimension. In the example, the same levels are being used in different ways.

```
DIMENSION TIME hierarchies=(Year_Months Year_Weeks);

       HIERARCHY Year_Months levels=(year month day);
       HIERARCHY Year_Quarter levels=(year quarter day);

       LEVEL year type=YEARS caption='Year';
       LEVEL year type=QUARTER caption='Quarter';
       LEVEL month type=MONTHS caption='Month';
       LEVEL day type=DAYS caption='Day';
```

Required Arguments

dim-name
names a dimension by using a valid SAS name up to 32 characters. For naming guidelines, see Naming Guidelines for SAS OLAP Server"Naming Guidelines for SAS OLAP Server" on page 120

HIERARCHIES=*(hier-name...hier-nameN)*
specifies the name of one or more hierarchies as defined by HIERARCHY statements.

Note: If you are building a cube that will contain multiple national languages, then DIMTABLELIBREF= and DIMTABLEMEMPREF= are required instead of DIMTBL=. △

Options

DESC | DESCRIPTION='*string*'
specifies any number of characters that can be used to create a meaningful description of the dimension. Third-party applications that report on cube data might display this description. If the text includes blank spaces or any characters that are not permitted in a valid SAS name, then enclose the text within quotation marks.

Default: *dim-name*

CAPTION='*string*'
specifies a maximum of 256 characters that can be used to create a meaningful description of the dimension. Third-party applications that report on cube data might display this description. If the text includes blank spaces or any characters that are not permitted in a valid SAS name, then enclose the text within quotation marks.

Default: *dim-name*

TYPE=TIME
identifies the dimension as a TIME dimension.

Requirement: You must set this option for a TIME dimension. TIME is the only valid value for this option.

Interaction: You can use LEVEL statements to specify the time period of each level in the TIME dimension.

SORT_ORDER=ASCENDING | DESCENDING | ASCFORMATTED | DESFORMATTED | DSORDER
specifies a sort order for all levels in the dimension. Values that are returned from queries display in this order.

Default: ASCENDING

Interaction: This setting is overridden if sort order is set in a LEVEL statement.

Tip: To specify a sort order for each level within a dimension, set the SORT_ORDER= option in each LEVEL statement. Values that are returned from queries display in this order.

DIMTBL=*libname.memname*
specifies the valid, two-level SAS name for a dimension table in the star schema that is specified with the FACT= option in the PROC OLAP statement. The dimension table must contain one column for each dimension level name (specified with the LEVELS= option in HIERARCHY statements) and one column for the dimension key. However, if the dimension key is also a level, then the dimension table needs to have only as many columns as there are levels in the dimension. Member metadata for the dimension is derived from the information in the level columns of the dimension table.

You can also specify data set options with DIMTBL=. Options are stored within the cube and reapplied when the data is accessed at run time. For more information, see "Data Set Options" in *SAS Language Reference: Concepts*.

Note: The fact table does not have to contain all of the members. However, the fact table cannot contain any members that are not described by the level columns. △

Note: The same dimension tables can be used to load cubes that have some, but not all, dimensions in common. This means that it is possible for multiple cubes to share the same dimension *data*. △

Note: If you are building a cube that will contain multiple national languages, then replace the DIMTBL= option with DIMTABLELIBREF= and DIMTABLEMEMPREF= options. In addition, you must create a USER_DEFINED_TRANSLATIONS statement. △

DIMKEY=*dimension-table-column*
specifies the name of the column in the dimension table that is specified in the DIMTBL= option. That column must contain values that correspond to fact key values in the fact table and be a value that corresponds to a unique combination of level values in the fact table.

Note: The corresponding fact key is specified with the FACTKEY= option. The fact table is specified with the FACT= option in the PROC OLAP statement. △

For example, for a dimension that is composed of three levels—**NAME**, **ADDRESS**, and **INCOME**—a dimension key named **CUSTOMER_ID** might exist. In this dimension, each unique value of **CUSTOMER_ID** corresponds to a unique combination of **NAME**, **ADDRESS**, and **INCOME**.

Table A1.3 Sample Dimension Data That Illustrates How Unique DIMKEY Values Correspond to Unique Combinations of Level Values

CUSTOMER_ID	NAME	ADDRESS	INCOME
1	Juan	hostel	2000
2	Shelly	apartment	2000
3	Paul	house	25000
4	Makoto	castle	250000000

FACTKEY=*fact-table-column*
specifies the name of the column in the fact table that corresponds to the dimension table column that is specified with the DIMKEY= option. The name does not have to match the DIMKEY name. Referring back to the previously discussed example, the FACTKEY name could be **CUST_NO** even though the DIMKEY name is **CUSTOMER_ID**. However, even if the names are different, the underlying data must match. For example, you must match numeric columns with numeric columns and character columns with character columns. In addition, if the FACTKEY is a character column, then it must be the same length as the DIMKEY column. If the FACTKEY is a numeric column, then it is handled as a decimal precision number (rather than as an integer).

DIMTABLELIBREF=
specifies the library for the data sets that exist, for this dimension, in each language that is specified by the USER_DEFINED_TRANSLATIONS statement. The library is associated with the dimension and not the language. You cannot put different languages in different libraries, but you can put different dimensions in different libraries. This option is required if you are using the Multiple Language Support capabilities of the SAS OLAP Server. It is also used in conjunction with the DIMTABLEMEMPREF= option.

DIMTABLEMEMPREF=
specifies the member prefix for the translated dimension tables. The member prefix is the prefix of the data set name. The suffix of the name is provided by the USER_DEFINED_TRANSLATIONS statement. For example, if the member prefix is **dealdim_** and the suffix is **da_DK**, then PROC OLAP looks for a data set named **dealdim_da_DK.sas7bdat** in the library that is specified by the DIMTABLELIBREF= option. DIMTABLEMEMPREF= is required if you are using

the Multiple Language Support capabilities of the SAS OLAP Server. It is used in conjunction with the DIMTABLELIBREF= option and the USER_DEFINED_TRANSLATIONS statement. This option follows the naming guidlines for SAS OLAP server"Naming Guidelines for SAS OLAP Server" on page 120.

LEVEL Statement

The LEVEL statement provides additional information about a level specified with the LEVELS= option in a HIERARCHY statement, and enables you to set options for ragged hierarchies.

LEVEL *level-name <option(s)>*;

For TIME dimensions, you can use LEVEL statements to specify a time period for each level in the dimension. However, if you specify the time period for one level, then you must specify the time period for all levels. You also use LEVEL statements to supply information such as a level description or a level-specific sort order. You can have a maximum of 256 levels per cube.

Note: Levels that are shared between hierarchies share the same ragged hierarchy option settings. The options are EMPTY_CHAR=, EMPTY_NUM=, EMPTY=, and IGNORE_EMPTY. △

Note: Levels use formats as specified in the input data source. To override the format, you can use a SAS FORMAT statement. △

Note: When you rebuild a cube that has been physically deleted, the rebuilt cube still uses the formats that were originally used to build the cube and were saved in the cube's metadata. This means that the rebuilt cube does not automatically include any formatting changes that you might have made in the input data source. To manually specify the new formats, edit and rebuild the cube by using SAS OLAP Cube Studio. △

Required Arguments

level-name
specifies a valid SAS name for the level that matches the name of a corresponding column in the input data. (You can use a column as a level even if it is also being used as a measure.) This is the same name that is used in the LEVELS= option in the HIERARCHY statement. Level names must be unique within a cube. For naming guidelines, see Naming Guidelines for SAS OLAP Server"Naming Guidelines for SAS OLAP Server" on page 120

Options

DESC | DESCRIPTION='*string*'
specifies any number of characters that can be used to create a meaningful description of the level. Third-party applications that report on cube data might display this description. If the text includes blank spaces or any characters that are not permitted in a valid SAS name, then enclose the text within quotation marks.

Default: The value of the CAPTION= option if one exists; otherwise, the column's label. If there is no label available, the default is the level name.

CAPTION='*string*'
specifies a maximum of 256 characters that can be used to create a meaningful description of the level. Third-party applications that report on cube data might display this description. If the text includes blank spaces or special characters that are not permitted in a valid SAS name, then enclose the caption within quotation marks.

Default: The column's label in the input data source. If there is no label available, the default is the level name.

TYPE=YEAR | HALF_YEARS | QUARTERS | MONTHS | WEEKS | DAYS | HOURS | MINUTES | SECONDS
if you specify the TYPE=TIME option in the DIMENSION statement, then you can use this LEVEL statement option to specify the time period for the dimension levels.

Requirement: If you specify a time period for one level in the TIME dimension, then you must specify the time period for all levels in the dimension. With regard to drill path, identify the levels from the most general time period to the most specific.

SORT_ORDER=ASCENDING | DESCENDING | ASCFORMATTED | DESFORMATTED | DSORDER
specifies a sort order for a level within a dimension. Values that are returned from queries display in this order.

Default: If a sort order is not specified in the DIMENSION statement or in the LEVEL statement, then the default order of ASCENDING is applied.

Interaction: This setting overrides the SORT_ORDER= setting in the DIMENSION statement.

EMPTY='*string*'
specifies that PROC OLAP should override any ragged hierarchy option settings in the PROC OLAP or HIERARCHY statements and instead use the specified string to identify missing hierarchy members. For example, if you set **`EMPTY='absent'`**, then hierarchy members that have the value **`'absent'`** are treated as missing for all levels in which the hierarchy appears. If you enclose the string within quotation marks, then you can use a maximum of 256 characters. If you do not use quotation marks, then the string must be a valid SAS name. For more information, see Naming Guidelines for SAS OLAP Server"Naming Guidelines for SAS OLAP Server" on page 120.

IGNORE_EMPTY
when this option is specified, then ragged hierarchy options in PROC OLAP or HIERARCHY statements are ignored for this level.

PROPERTY Statement

The PROPERTY statement assigns properties to specific levels within specified hierarchies.

PROPERTY *prop-name* LEVEL=*level-name* <*option(s)*>;

Each level can have more than one property assigned to it by using multiple PROPERTY statements. Property names must match the name of a column in the input data source, or you must use the COLUMN= option to specify the column name.

In the following example, the COLUMN= option is used in the first two PROPERTY statements because the column name is different from the property name. In this way,

the property named **Population** can be assigned to both the **country** level and the **state** level in the **geo** hierarchy. The level **state** has two properties: **Population** and **West_of_Miss**.

```
PROPERTY Population
         column=p_country
         hierarchy=geo
         level=country;
PROPERTY Population
         column=p_state
         hierarchy=geo
         level=state;
PROPERTY West_of_Miss
         hierarchy=geo
         level=state;
```

Required Arguments

prop-name
specifies a valid SAS name for the property. Usually this is the name of a column in the input data source. If it is not the name of a column, then you must include the COLUMN= option to specify the column name. For naming guidelines, see Naming Guidelines for SAS OLAP Server"Naming Guidelines for SAS OLAP Server" on page 120

LEVEL=*level-name*
specifies the name of the level that you are assigning the property to.

Options

HIERARCHY=(*hier-name ... hier-nameN*)
specifies the name of one or more hierarchies that contain the level. If you do not include the HIERARCHY option, then the property is automatically assigned to all occurrences of the level in all of the hierarchies in which it appears; otherwise, the property is assigned to the level only in the specified hierarchies.

COLUMN=column-name
specifies the name of a column from the input data source. You must use this option if the column name is not the same as the property name.

DESC | DESCRIPTION='*string*'
specifies any number of characters that can be used to create a meaningful description of the level. Third-party applications that report on cube data might display this description. If the description includes blank spaces or any characters that are not permitted in a valid SAS name, then enclose the text within quotation marks.

Default: The value of the CAPTION= option if one exists; otherwise, the column's label.

CAPTION='*string*'
specifies a maximum of 256 characters that can be used to create a meaningful description of the level. Third-party applications that report on cube data might display this description. If the caption includes blank spaces or special characters that are not permitted in a valid SAS name, then enclose the caption within quotation marks.

Default: The column's label if it exists, otherwise the property name.

HIERARCHY Statement

The HIERARCHY statement specifies the navigational order of the levels in a dimension.

HIERARCHY *hier-name* LEVELS=(*level–name1 <level–name2 ...level-nameN>*) *<option(s)>*;

You must have at least one HIERARCHY statement for each DIMENSION statement; however, you can also have multiple hierarchies per dimension. Levels in the same dimension can be shared between hierarchies. Every level in a dimension must be assigned to a hierarchy in the dimension. You can have a maximum of 19 levels per hierarchy. There is no limit to the number of hierarchies per dimension.

Following is an example of a HIERARCHY statement that specifies three levels:

```
HIERARCHY Geography levels=(country region division);
```

Required Arguments

hier-name
specifies a valid SAS name for the hierarchy. This name is also used in the HIERARCHIES= option in the DIMENSION statement. The *hier-name* cannot be the same as any of its level names. Hierarchy names must be unique within the cube. If the hierarchy that you are defining is the only one in the dimension, then the hierarchy name must match the dimension name. For other naming guidelines, see Naming Guidelines for SAS OLAP Server"Naming Guidelines for SAS OLAP Server" on page 120.

LEVELS=(*level-name1 <level-name2 ...level-nameN>*)
specifies a valid SAS name for at least one level. These names correspond to columns in your input data and are used in any optional LEVEL statements. Level names must be unique within a cube and cannot be the same as the *hier-name*. (You can use a column as a level even if it is also being used as a measure.) Enter one or more names, separated by a space. Enter the level names in the order in which you want them to be used, beginning with the top level. For naming guidelines, see Naming Guidelines for SAS OLAP Server"Naming Guidelines for SAS OLAP Server" on page 120.

Requirement: If the hierarchy is part of a TIME dimension, then the levels must be listed in order from most general to least general based on their assigned TYPE. For example, a TYPE=YEAR level must be listed before a TYPE=QUARTER level.

Options

DESC | DESCRIPTION='*string*'
specifies any number of characters that can be used to create a meaningful description of the hierarchy. Third-party applications that report on cube data might display this description. If the text includes blank spaces or any characters that are not permitted in a valid SAS name, then enclose the text within quotation marks.

Default: *the hierarchy caption, which may be the default, hier-name.*

CAPTION='*string*'
specifies a maximum of 256 characters that can be used to create a meaningful description of the hierarchy. Third-party applications that report on cube data might display this description. If the text includes blank spaces or any characters that are not permitted in a valid SAS name, then enclose the text within quotation marks.

Default: *hier-name*

EMPTY_CHAR='*string*'
for ragged hierarchies, this option specifies the string that identifies missing characters as hierarchy members. For example, by default, SAS identifies missing characters as a blank space. To identify blank spaces as missing hierarchy members, enter `''`as the value of the EMPTY_CHAR= option. To identify every instance of the word "empty" as missing hierarchy members, enter **`'empty'`**. In this case, hierarchy members that have the value **`'empty'`** are treated as missing for all levels in which the hierarchy appears and they are ignored when you drill down into the cube. If you enclose the string within quotation marks, then you can use a maximum of 256 characters. If you do not use quotation marks, then the string must be a valid SAS name. For more information, see Naming Guidelines for SAS OLAP Server"Naming Guidelines for SAS OLAP Server" on page 120.

Interaction: This option overrides any EMPTY_CHAR option that is specified in the PROC OLAP statement.

EMPTY_NUM='*string*'
for ragged hierarchies, this option specifies the string that identifies missing numeric values as hierarchy members. For example, by default, SAS identifies missing numeric values as a single period (.). To identify periods as missing hierarchy members, enter **`'.'`** as the value of the EMPTY_NUM= option. To identify every instance of the value "123" as missing hierarchy members, enter **`'123'`**. In this case, hierarchy members that have the value **`'123'`** are treated as missing for all levels in which the hierarchy appears and they are ignored when you drill down into the cube. If you enclose the string within quotation marks, then you can use a maximum of 256 characters. If you do not use quotation marks, then the string must be a valid SAS name. For more information, see Naming Guidelines for SAS OLAP Server"Naming Guidelines for SAS OLAP Server" on page 120.

Note: If there is no format associated with the member value, then **`BEST12`** is used as the format. △

Interaction: This option overrides any EMPTY_NUM option set in the PROC OLAP statement. In addition, this option is ignored if the IGNORE_EMPTY option is set in the HIERARCHY statement or the corresponding LEVEL statement.

IGNORE_EMPTY
specifies that PROC OLAP should ignore any ragged and unbalanced hierarchy option settings in the PROC OLAP statement.

DEFAULT
identifies a hierarchy as the default hierarchy for the dimension that is defined by the DIMENSION statement.

Default: The first hierarchy listed for the dimension

MEASURE Statement

The MEASURE statement defines the cube's measures and indicates how they map to the input data.

MEASURE *measure-name* STAT=*statname* <*option(s)*>;

Include one MEASURE statement for each measure in the cube. Each cube must have at least one measure. Measure names must be unique. You can have a maximum of 1,024 measures per cube.

Here are two examples of the MEASURE statement:

```
MEASURE Sales_Sum
        STAT=SUM
        COLUMN=sales
        AGGR_COLUMN=sales
        DESC='Sales Summary'
        UNITS='Dollars'
        FORMAT=COMMA9.2;


MEASURE Sales_Min
        STAT=MIN
        COLUMN=sales
        AGGR_COLUMN=isales
        DESC='Sales Min'
        UNITS='Dollars'
        FORMAT=COMMA9.2;
```

Note: All cube aggregations have identical measures. △

Required Arguments

measure-name
specifies a valid SAS name for the measure. The name must be unique. For naming guidelines, see Naming Guidelines for SAS OLAP Server"Naming Guidelines for SAS OLAP Server" on page 120.

STAT= *statname*
specifies the statistic for the measure. The following base statistics are available: N, NMISS, SUM, MAX, MIN, or USS. In addition, these derived statistics are also available: AVG, RANGE, CSS, VAR, STD, STDERR, CV, T, PRT, LCLM, or UCLM.

New cubes that are based on a data source that contains *existing summarized data* (where such data has been indicated in at least one AGGREGATION statement via the TABLE= option), must include measure statements for the stored statistics required for each derived statistic that you want to create for the new cube. For example, if you want to calculate AVG, you must create measures for N and SUM, as well as AVG. The following table indicates which stored statistics are required for each derived statistic:

Table A1.4 Stored Statistics Required for Each Derived Statistic

Derived Statistics	Required Stored Statistics
AVG	N, SUM
CSS	N, SUM, USS

Derived Statistics	Required Stored Statistics
RANGE	MIN, MAX
VAR, STD, STDERR, CV, T, PRT, LCLM, UCLM	N, SUM, USS

Note: For information about statistic formulas, see "Keywords and Formulas" in *Base SAS Procedures Guide.* △

For cubes that are *not* loaded from a fully summarized data source (that is, you specified a data source by using the DATA | FACT= option), some statistics use formats taken from the input data source. Specifically, if the statistic is SUM, MIN, MAX, RANGE, AVG, STD, STDERR, LCLM, or UCLM, then PROC OLAP uses the format that is assigned to the column specified by the COLUMN | ANALYSIS= option. The following table lists the formats used for the other supported statistics:

Table A1.5 Default Formats Used for Statistics

Statistic	Format Used
CSS	BEST.
CV	8.2
N	12.0
NMISS	10.0
PRT	6.4
T	7.3
USS	BEST.
VAR	BEST.

For cubes that *are* loaded from a fully summarized data source (that is, you specified the data source by using the AGGREGATION statement), the default format is BEST12.

To override the default formats, you can either set the FORMAT= option or use a SAS FORMAT statement.

Note: The FORMAT= option also overrides a FORMAT statement. △

Note: When you rebuild a cube that has been physically deleted, the rebuilt cube still uses the formats originally saved in the cube's metadata. This means that the rebuilt cube does not automatically include any formatting changes that you might have made in the input data source. To manually specify the new formats, edit and rebuild the cube by using SAS OLAP Cube Studio. △

COLUMN | ANALYSIS=*anlvar*

specifies the name of a numeric column that is contained in the cube's input data source. (You can use a column as a measure even if it is also being used as a level.)

If the cube is based on an *unsummarized* data source, then *anlvar* is the name of the column in that data source from which the measure will be calculated. Use COLUMN= to specify the column.

If the cube is based on a *summarized* data source, then *anlvar* can be the name of the numeric column in the data source that was used as the analysis variable for the pre-calculated measure. It can also be a name that identifies a logical association between measures with the same *anlvar* name. For example, if your cube has three measures, N, SUM, and AVERAGE, and those measures were

derived from the same analysis variable, then you could specify **ANALYSIS=Sales** to logically link the three measures through their shared analysis variable. You would also identify the analysis variable in the AGGR_COLUMN= option.

If the cube consists of a combination of summarized and unsummarized data sources, then *anlvar* refers to both a physical and a logical entity. For example, you might have a cube that requires a physical analysis variable to create a crossing but that same cube already contains other, higher level aggregations. In this case, the analysis variable is also used to logically link the measures in the pre-existing aggregations that were derived from the same input column. You would also identify the analysis variable in the AGGR_COLUMN= option.

Default: *measure-name*

Interaction: An unsummarized data source is specified with the DATA | FACT= option in the PROC OLAP statement. A summarized data source is specified with the TABLE= option in an AGGREGATION statement.

Options

DESC | DESCRIPTION='*string*'
specifies any number of characters that can be used to create a meaningful description of the measure. Third-party applications that report on cube data might display this description. If the text includes blank spaces or any characters that are not permitted in a valid SAS name, then enclose the text within quotation marks.

Default: *measure-name*

CAPTION='*string*'
specifies a maximum of 256 characters that can be used to create a meaningful description of the measure. Third-party applications that report on cube data might display this description. If the text includes blank spaces or any characters that are not permitted in a valid SAS name, then enclose the text within quotation marks.

Default: The default is based on the statistic and the COLUMN= value, as shown in the following table. For example, if the statistic is **SUM** and the COLUMN= value is **Sales**, then the default caption is **Sum of Sales**.

Table A1.6 Defaults for the CAPTION= Option If No Caption Is Specified

Statistic Used for Measure	Default Caption
AVG	Average *measure-column-name*
CSS	Corrected Sum of Squares of *measure-column-name*
CV	*Measure-column-name* Coefficient of Variation
LCLM	*Measure-column-name* Lower Confidence Limit
MAX	Maximum *measure-column-name*
MIN	Minimum *measure-column-name*
N	Number of Values for *measure-column-name*

Statistic Used for Measure	Default Caption
NMISS	Number of Missing Values for *measure-column-name*
PRT	Probability of Greater Absolute Value for *measure-column-name*
RANGE	*Measure-column-name* Range
STD	*Measure-column-name* Standard Deviation
STDERR	*Measure-column-name* Standard Error of Mean
SUM	Sum of *measure-column-name*
T	*Measure-column-name* T Value
UCLM	*Measure-column-name* Upper Confidence Limit
USS	*Measure-column-name* Uncorrected Sum of Squares
VAR	*Measure-column-name* Variance

AGGR_COLUMN=*input-column*
specifies the name of the numeric column in the summarized input data that contains the values for the measure. The source of the summarized input data is specified in the AGGREGATION statement. This option is valid only for stored statistics.
Default: *measure-name*

UNITS='*string*'
specifies a maximum of 256 characters that can be used to create a meaningful description of the measure's units (for example, "pounds sterling"). Third-party applications that report on cube data might display this description. If the text includes blank spaces, mixed-case letters, special characters, then enclose the text within quotation marks.

FORMAT=*sas-format-name*
specifies the SAS format to be used to display the value of the measure. This format overrides the default format (see STAT= for more information) and any format that is specified in a SAS FORMAT statement.

Note: When you rebuild a cube that has been physically deleted, the rebuilt cube still uses the formats that were originally saved in the cube's metadata. This means that the rebuilt cube does not automatically include any formatting changes that you might have made in the input data source. To manually specify the new formats, edit and rebuild the cube by using SAS OLAP Cube Studio. △

DEFAULT
identifies a measure as the default measure for the cube.
Default: The measure defined in the first MEASURE statement

AGGREGATION Statement

The AGGREGATION statement defines an aggregation of the cube based on level information that you provide.

AGGREGATION *level-name1 <level-name2 level-name3 ... level-nameN>* < / TABLE=*libname.dataset* ></ NAME=' *aggregation-name*'>;

You can specify level names that are associated with an unsummarized data source, or you can specify level names that match columns in a table that contains existing aggregated data. The levels can exist in more than one dimension. You do not need to include dimension names because level names must be unique across dimensions

Here is an example of an AGGREGATION statement that specifies three levels and uses the / NAME= option:

```
AGGREGATION country prodtype year /name='Product Types by Country';
```

Required Arguments

level-name1 <level-name2 level-name3 ...level-nameN>

is the name of one or more levels to be used to create the aggregation. You do not have to include all levels that are specified in all HIERARCHY statements, but the names that you do specify must match the names that are used in the HIERARCHY statements. You can include a TABLE= option to identify a table that contains existing aggregated information for your specified levels. The levels that you specify must match columns in the input table.

Restriction: Levels must be listed in drill-path order. In addition, levels must be contiguous within a hierarchy. You cannot specify an aggregation that contains a summary level that could never be requested. For example, if your TIME hierarchy contains the level **Year**, **Month**, and **Day**, you could specify **Year** and **Month** as an aggregation but not **Month** by itself.

Options

Note: For information about options that can be used to optimize cube creation and query performance, see Syntax Options Used for Performance. △

/ TABLE=*libname.dataset*

specifies the name of a SAS data set or data view that contains the data for one aggregation. Every level that is listed in the AGGREGATION statement must match a column that contains aggregation information in the specified table. Place this option after the list of level names.

Analysis columns in the table are mapped to the numeric columns that are specified with the AGGR_COLUMN= option in MEASURE statements.

You can also set data set options with /TABLE=. Options are stored within the cube and reapplied when the data is accessed at run time. For more information, see "Data Set Options" in *SAS Language Reference: Concepts*.

Restriction: You cannot use the /TABLE= option in an AGGREGATION statement that is used to add an aggregation to an existing cube.

/ NAME='*aggregation-name*'

specifies a maximum of 256 characters as the name of the aggregation. If the name includes blank spaces or any characters that are not permitted in a valid SAS name, then enclose the name within quotation marks. The name is stored with the cube's metadata. Place this option after the list of level names. An example is **AGGREGATION Year Month Product / NAME= 'Year Prod';.**

Default: A name assigned by SAS such as **AGGR1**

Requirement: You must include a / before the TABLE= and NAME= options; however, if you use both options, you include only one slash. An example is

```
AGGREGATION Year Month Product / TABLE=financial.products
NAME='Year Prod';.
```

DATAPATH=('*path-name*' ...'*pathnameN*')
specifies the location of one or more partitions (.DPF files) in which to place aggregation table data. The data is distributed by cycling through each partition location according to the partition size. This is set by using the PARTSIZE= option. For example, if you specify **DATAPATH=('c:\data1' 'd:\data2')**, then PROC OLAP places the first partition of the aggregation table into directory **c:\data1**, the second partition of the table into directory **d:\data2**, the third partition of the table into **c:\data1**, and so on. It is also possible to have aggregation tables that use fewer than the specified number of partitions. For example, your aggregation table might fit entirely into **c:\data1**.

Default: The cube subdirectory of the location that is specified by the PATH= option in the PROC OLAP statement

INDEXPATH=('*path-name*' ...'*pathnameN*')
specifies the locations of the index component files (.IDX and .HYB files) that correspond to each aggregation table partition as specified by the DATAPATH= option.

Default: The cube subdirectory of the location that is specified by the PATH= option in the PROC OLAP statement

COMPRESS | NOCOMPRESS
specifies whether or not to store the aggregation table in a compressed format on disk.

Default: NOCOMPRESS

INDEX | NOINDEX
specifies whether or not to create the specified aggregation with indexes. For faster cube creation and updates, you can set this option to NOINDEX; however, the lack of indexes might adversely affect query performance.

Note: Indexes are not created for aggregations that have fewer than 1,024 records. △

Default: INDEX

PARTSIZE=*size-in-megabytes*
specifies the partition size in megabytes of the aggregation table partitions (.DPF files) and their corresponding index components (.IDX and .HYB files).

Default: 128 megabytes. The minimum value is 16.

SEGSIZE=*number-of-rows-in-kb*
specifies the number of observations (table rows) in kilobytes to include in the file segment of the index component. The minimum size is 1k (1,024 rows), so the value of SEGSIZE= is a multiple of 1024 as expressed in kilobytes. The segmented indexes are used to optimize WHERE-expression processing. Each parallel thread is given a segment of the table to evaluate that is equal to the SEGSIZE= value.

Default: 8 kilobytes (8,192 rows). The minimum size is 1 kilobyte (1,024 rows).

DROP_AGGREGATION Statement

The DROP_AGGREGATION statement removes an aggregation from the specified cube.

DROP_AGGREGATION *level-name1 < level-name2 ... level-nameN >* / NAME=*aggregation-name* ;

You can specify the levels that are in the aggregation, or the name of the aggregation, or both the levels and the name.

Required Arguments

At least one of the following arguments is required for a DROP_AGGREGATION statement:

level-name1 <level-name2 ... level-nameN>
specifies the names of the levels that are in the aggregation that you want to drop. An example is **DROP_AGGREGATION Year Month Product.**

/ NAME='*aggregation-name*'
specifies the name of the aggregation that you want to drop. If the name includes blank spaces or any characters that are not permitted in a valid SAS name, then enclose the name within quotation marks. An example is **DROP_AGGREGATION NAME=''Year Prod''**.

Requirement: You must include a / before NAME= if you specify both a list of levels and a name. An example is **DROP_AGGREGATION Year Month Product / NAME='Year Prod'.**

DEFINE Statement

The DEFINE statement defines a global calculated member or a named set for any cube that is registered in the SAS Metadata Repository.

DEFINE MEMBER | SET '*member-or-set-name*' AS '*mdx-expression*' ;

A calculated member is a dimension member that has been calculated from the member values in the input table. Only the definition of the member is stored; the value is calculated when a query is submitted. A named set is an alias for a specified MDX expression. Named sets are often used to make complex MDX queries easier to read and maintain.

The defined calculated members and named sets are available to any session that creates a query in the context of the OLAP Server and the schema defined in the METASRV statement of the proc olap script used to create the global member or set.

DEFINE statements can apply to more than one cube, so the CUBE= option is not required to use this statement. The METASVR statement verifies that the cube definition exists in the metadata repository.

The DEFINE statement can be used alone as shown in this example, which defines two calculated members and one named set. The METASVR is the only other required statement. To define multiple sets or cubes, separate option values with a comma.

```
PROC OLAP;
METASVR OLAP_SCHEMA='Services Schema'
        REPOSITORY='services'
        HOST='misdept.us.mar.com'
        PORT=9999
        PROTOCOL=com
        USERID=jjones
```

```
        PW='my password';
DEFINE member '[mddbcars].[Measures].[avg]' AS
              '[Measures].[sales_sum]/[Measures].[sales_n]',
       member '[sales].[Measures].[stat1]' AS
              '[Measures].[qty] +1',
       set '[campaign].[myset]' AS
           '[campaign_dates].[All compaign_dates].children';
run;
```

The DEFINE statement can also be used with a PROC OLAP program that creates a cube or with a program that adds aggregations to or deletes aggregations from an existing cube. Cube builds, additions, and deletions occur before the DEFINE statement is processed, so the DEFINE statement is not processed if those statements fail.

```
PROC OLAP DATA=olapsio.cars CUBE=mddbcars PATH='d:\services\';

METASVR OLAP_SCHEMA='Services Schema'
        REPOSITORY='cars'
        HOST='misdept.us.mar.com'
        PORT=9999
        PROTOCOL=com
        USERID=jjones
        PW='my password';

DIMENSION date HIERARCHIES=(date) SORT_ORDER=ASCENDING;
HIERARCHY date LEVELS=(dte);
LEVEL dte;

DIMENSION cars HIERARCHIES=(cars) SORT_ORDER=ASCENDING;
HIERARCHY cars LEVELS=(car color);

DIMENSION dealers HIERARCHIES=(dealers) SORT_ORDER=ASCENDING;
HIERARCHY dealers LEVELS=(dealer dest);

MEASURE sales_sum COLUMN=SALES STAT=sum FORMAT=dollar15.2;
MEASURE sales_n COLUMN=SALES STAT=n FORMAT=12.0;

DEFINE member '[mddbcars].[Measures].[avg]' AS
              '[Measures].[sales_sum] / [Measures].[sales_n]';

run;
```

Required Arguments

MEMBER | SET
: indicates whether you are creating a calculated member or a named set.

'*member-or-set-name*'
: specifies the name of the member or set that you are creating. If you are creating a calculated member, then this value specifies a name for the member that will be calculated by the MDX expression. If you are creating a named set, then this value is the alias for the specified MDX expression.

AS *'mdx-expression'*
specifies the MDX expression.

USER_DEFINED_TRANSLATIONS Statement

The USER_DEFINED_TRANSLATIONS statement is required to use the Multiple Language Support capabilities of the SAS OLAP Server. This statement specifies the locales that are associated with the data sets that you specify in the DIMENSION statement.

USER_DEFINED_TRANSLATIONS one or more of 56 locales ;

Note: Alternative statement names are UDT and USER_DEFINED_TRANSLATION. △

PROC OLAP uses the UDT statement information, along with DIMENSION statement options, to read your alternate locale data sets and create locale-specific metadata for use at query time. Query results are returned in the language of the requested locale. The Multiple Language Support feature is available only for cubes that are loaded from a star schema. The alternate locale data set names consist of a prefix, which indicates the member, and a suffix, which indicates the language. The DEFINE statement supplies the suffix. The DIMTABLEMEMPREF= option in the DIMENSION statement specifies the member prefix. For example, if the member prefix is **dealdim_** and the suffix is **pl_PL**, then PROC OLAP looks for a data set named **dealdim_pl_PL.sas7bdat** in the library that is specified by the DIMTABLELIBREF= option.

The following sample code looks for these dimension data sets in the **mylib** library. The default locale is the first locale specified in the UDT statement. Additionally, the default locale does not use the suffix that is defined by the UDT statement. In this example, Polish is the default locale, so the suffix is not used.

Table A1.7 Locales and Associated Data Set Names

Locale	Dimension Data Sets		
English	ctimedim_en_US	cardim_en_US	dealdim_en_US
Japanese	ctimedim_ja_JP	cardim_ja_JP	dealdim_ja_JP
Polish	ctimedim_	cardim_	dealdim_

```
DIMENSION date hierarchies=(date) sort_order=ASCENDING
               dimtablelibref=mylib
               dimtablemempref=ctimedim_
               factkey=dte
               dimkey=dte;
HIERARCHY date levels=(dte);
LEVEL dte;
DIMENSION cars hierarchies=(cars) sort_order=ASCENDING
               dimtablelibref=mylib
               dimtablemempref=cardim_
               factkey=carkey
               dimkey=carkey;
```

```
DIMENSION cars levels=(car color);
DIMENSION dealers hierarchies=(dealers) sort_order=ASCENDING
                  dimtablelibref=mylib
                  dimtablempref=dealdim_
                  factkey=dealerkey
                  dimkey=dealerkey;
HIERARCHY dealers levels=(dealer dest);

USER_DEFINED_TRANSLATIONS pl_PL  en_US  ja_JP;
```

Required Argument

one or more of 56 locales
specifies the locales that correspond to the data sets contained in the library that is specified by the DIMTABLELIBREF= option in the DIMENSION statement. Separate locales with a space. Your choices are listed below:

ar_AE=Arabic (United Arab Emirates)

bg_BG=Bulgarian (Bulgaria)

be_BY=Byelorussian (Belarus)

zh_CN=Chinese (China)

zh_HK=Chinese (HongKong)

zh_MO=Chinese (Macau)

zh_SG=Chinese (Singapore)

zh_TW=Chinese (Taiwan)

hr_HR=Croatian (Crotia)

cs_CZ=Czech (Czech Republic)

da_DK=Danish (Denmark)

nl_NL=Dutch (Netherlands)

en_AU=English (Australia)

en_CA=English (Canada)

en_CB=English
(Caribbean)

en_IE=English
(Ireland)

en_JM=English
(Jamaica)

en_NZ=English
(New Zealand)

en_ZA=English
(South Africa)

en_GB=English
(United Kingdom)

en_US=English
(United States)

et_EE=Estonian
(Estonia)

fi_FI=Finnish
(Finland)

fr_BE=French
(Belgium)

fr_CA=French
(Canada)

fr_FR=French
(France)

fr_CH=French
(Switzerland)

de_AT=German
(Austria)

de_DE=German
(Germany)

de_CH=German
(Switzerland)

el_GR=Greek
(Greece)

he_IL=Hebrew
(Israel)

hu_HU=Hungarian
(Hungary)

is_IS=Icelandic
(Iceland)

it_IT=Italian (Italy)

it_CH=Italian
(Switzerland)

ja_JP=Japanese
(Japan)

ko_KR=Korean
(South Korea)

lv_LV=Latvian
(Lettish) (Latvia)

lt_LT=Lithuanian
(Lithuania)

no_NO=Norwegian
(Norway)

pl_PL=Polish
(Poland)

pt_BR=Portuguese
(Brazil)

pt_PT=Portuguese
(Portugal)

ro_RO=Romanian
(Romania)

ru_RU=Russian
(Russia)

sr_YU=Serbian
(Yugoslavia)

sk_SK=Slovak
(Slovakia)

sl_SI=Slovenian
(Slovenia)

es_MX=Spanish
(Mexico)

es_ES=Spanish
(Spain)

sv_SE=Swedish
(Sweden)

th_TH=Thai
(Thailand)

tr_TR=Turkish
(Turkey)

uk_UA=Ukrainian
(Ukraine)

vi_VN=Vietnamese
(Vietnam)

SAS Servers and Character Encoding

If your server metadata contains characters other than those typically found in English, then you must be careful to start your server with an encoding= or locale= system option that accommodates those characters. For example, a SAS server started

with the default US English locale cannot read metadata that contains Japanese characters. SAS will fail to start and log a message indicating a transcoding failure.

In general, different SAS jobs or servers can run different encodings (such as ASCII/ EBCDIC or various Asian DBCS encodings) as long as the encoding that is used by the particular job or server can represent all the characters of the data being processed. In the context of server start up, this requires that you review the characters used in the metadata describing your server (as indicated by the server= objectserverparm) to ensure that SAS runs under an encoding that supports those characters.

Tables Used to Define Cubes

There are five types of tables that can be used to define a cube:

- detail tables
- fact tables and dimension tables (for cubes that are based on star schemas)
- aggregation tables
- drill-through tables

Detail Tables

A detail, or base, table is any table that is defined in the SAS Metadata Repository that contains the columns for the measures and levels of a cube. A detail table consists of unsummarized data that must include one column for each level and one numeric analysis column for each set of measures that will be generated.

Fact Tables and Dimension Tables

A star schema refers to a set of input tables that are defined in the SAS Metadata Repository. A set of tables includes a single fact table and one or more dimension tables. A fact table must contain one numeric analysis column for each set of measures that will be generated. For levels, a fact table will either contain the columns for the levels of a dimension or contain a key column that links the fact table with a dimension table that contains the columns for the levels of a dimension.

The following statements are also true for star schemas:

- A fact table can contain a dimension. When this occurs, all the level columns are contained in the fact table and no fact or dimension key is required.
- If the dimension levels are defined in a dimension table, then all the level columns for that dimension must be contained in the same dimension table.
- Both the dimension keys and the fact keys are single columns, not combinations of columns.
- The dimension key can also be a level in the dimension.

Aggregation Tables

Aggregation tables are fully summarized, external relational tables. All aggregation tables must contain a column for each measure in the cube where the statistic for the measure is one of the following: N, NMISS, SUM, MAX, MIN, or USS. Columns for derived measures cannot be stored on the aggregation table and are ignored if they exist. Derived measures are always computed at query time. (See the STAT= option in the MEASURE statement for more information about stored and derived statistics.)

An aggregation table can be used in two ways:

- As an NWAY data source for the cube. In this case, the table must contain a column for every level in the cube and a column for every stored measure.

- As a subaggregation for the cube. In this case, the table must include a column for each level of the aggregation and a column for every stored measure.

Drill-Through Tables

Drill-through tables are views, data sets, or other data files maintained by the user that represent all of the relevant input data that is used to define a cube. This data is later accessed when performing drill-through actions from a client. PROC OLAP checks that the drill-through table has the correct level and measure names but never looks at the actual data contents of the table. The name of the drill-through table is stored in the cube's metadata in place of a detail or fact table name. (The detail or fact table name is still referred to when an action is performed on a cube.) The drill-through table name can be set either when the cube is first created or during an update.

Drill-through tables can be used by client applications to provide a view from processed data into the underlying data source.

Naming Guidelines for SAS OLAP Server

For SAS OLAP Server, names

- can be up to 32 characters in length.
- can contain mixed-case letters. SAS stores and writes the variable name in the same case that is used in the first reference to the variable. However, when SAS processes a variable name, SAS internally converts it to uppercase. You cannot, therefore, use the same variable name with a different combination of uppercase and lowercase letters to represent different variables. For example, cat, Cat, and CAT all represent the same variable.
- can contain characters other than Latin alphabet letters, numerals, and underscores, including embedded blanks. An exception to this is the dot character (.), which is an invalid character. You can do this by setting the SAS system option VALIDVARNAME=ANY. When this is set, PROC OLAP interprets the name as a *SAS name litteral*, which is a token that is expressed as a string within quotation marks, followed by the letter *n*. Here are some examples:

```
DIMENSION 'Product@Work Dimension'n hierarchies=('Product@Work Hierarchy'n);
HIERARCHY "Product@Work Hierarchy"n levels=(prodtype product);
```

The PROC saves the case of the FIRST LOGICAL ENCOUNTER of an object name in the PROC script and saves it as the final stored name in the cube metadata. Therefore, the logical order of statement processing is as follows:

DIMENSION

HIERARCHY

LEVEL

AGGREGATION

MEASURE

PROPERTY.

Every DIMENSION statement is processed before any HIERARCHY statement and every HIERARCHY statement is processed before any LEVEL statement. This rule affects hierarchy and level names only. The stored hierarchy names are copied from the DIMENSION statement and the stored level names are copied from the first HIERARCHY statement listed in the PROC that uses this level. Here is an example:

```
Hierarchy TimE levels= (Year Month Day);
Dimension Time hierarchies= (Time);
Level YEAR  type = year;
```

In this example the hierarchy name is stored in the cube as "Time" rather than "TimE". The level name for year is stored as "Year" rather than "YEAR". The AGGREGATION statement does not affect the case of the level names that are stored in the cube metadata.

Note: For further information about the VALIDVARNAME= system option, see "VALIDVARNAME=System Option" and "Names in the SAS Language" in *SAS Language Reference: Dictionary*. △

Loading Cubes

Loading Cubes from a Detail Table

The following table lists the PROC OLAP statements and options that you use to load a cube from a detail table. The detail table has a column for each level and at least one numeric analysis column from which one or more measures can be generated.

Table A1.8 Statements and Options Used for Loading Cubes from a Detail Table

Use these statements	Use these options	
AGGREGATION	The AGGREGATION statement is optional unless you are creating additional aggregations, in which case, you must specify the names of the contiguous levels to be used to create the aggregation. Use the / TABLE= option for cubes that are loaded from fully summarized tables.	
DIMENSION	HIERARCHIES=	Required
	DESC=	Optional
	CAPTION=	Optional
	TYPE=TIME	Required only for TIME dimensions
	SORT_ORDER=	Optional
HIERARCHY	LEVELS=	Required
	DESC=	Optional
	CAPTION=	Optional
LEVEL	DESC= CAPTION= TYPE=	
	The LEVEL statement is optional unless you want to specify time periods for each level in a TIME dimension. If you specify a time period for one level, then you must specify a time period for all levels. To specify a time period, you use the TYPE= option.	

Use these statements	Use these options	
MEASURE	STAT=	Required
	COLUMN | ANALYSIS=	Required
	AGGR_COLUMN	Required if you use the AGGREGATION statement with the /TABLE= option
	DESC=	Optional
	CAPTION=	Optional
	UNITS=	Optional
	FORMAT=	Optional
	DEFAULT=	Optional
METASVR	OLAP_SCHEMA=	Required
	REPOSITORY=	Optional
	HOST=	Optional
	PORT=	Optional
	PROTOCOL=	Optional
	USERID=	Optional
	PW=	Optional
PROC OLAP	DATA=	Required
	CUBE=	Required
	PATH=	Required
	DESC=	Optional
	NO_NWAY	Optional

Loading Cubes from a Star Schema

The following table lists the PROC OLAP statements and options that you use to load a cube from a star schema. A star schema is a set of input tables that are defined in a repository. The set of tables includes a single fact table and one or more dimension tables. The fact table must contain at least one numeric analysis column for each set of measures that will be generated.

Table A1.9 Statements and Options Used to Load Cubes from a Star Schema

Use these statements	Use these options	
PROC OLAP	FACT=	Required
	CUBE=	Required
	PATH=	Required
	DESC=	Optional
	NO_NWAY	Optional
METASVR	OLAP_SCHEMA=	Required

Use these statements	Use these options	
	REPOSITORY=	Optional
	HOST=	Optional
	PORT=	Optional
	PROTOCOL=	Optional
	USERID=	Optional
	PW=	Optional
DIMENSION	HIERARCHIES=	Required
	DESC=	Optional
	CAPTION=	Optional
	TYPE=TIME	Required only for TIME dimensions
	SORT_ORDER=	Optional
	DIMTBL=	Required for cubes that support one locale. If the cube will contain multiple national languages, replace this option with DIMTABLELIBREF= and DIMTABLEMEMPREF=.
	DIMKEY=	Required
	FACTKEY=	Required
	DIMTABLELIBREF=	Required if you are building a cube that will contain multiple national languages. Replaces DIMTBL=.
	DIMTABLEMEMPREF=	Required if you are building a cube that will contain multiple national languages. Replaces DIMTBL=.
LEVEL		The LEVEL statement is optional unless you want to specify time periods for each level in a TIME dimension. If you specify a time period for one level, then you must specify a time period for all levels. To specify a time period, you use the TYPE= option.
HIERARCHY	LEVELS=	Required
	DESC=	Optional
	CAPTION=	Optional
MEASURE	STAT=	Required
	COLUMN | ANALYSIS=	Required

Use these statements	Use these options	
	AGGR_COLUMN=	Required if you use the AGGREGATION statement with the /TABLE= option
	DESC=	Optional
	CAPTION=	Optional
	UNITS=	Optional
	FORMAT=	Optional
	DEFAULT=	Optional
AGGREGATION		The AGGREGATION statement is optional unless you are creating additional aggregations, in which case, you must specify the names of the contiguous levels to be used to create the aggregation. Use the /TABLE= option for cubes that contain aggregated data from tables other than the input data source.

Loading Cubes Using Summarized Data

The following table lists the PROC OLAP statements and options that you use to load cubes from a fully summarized data source (a crossing of all dimensions also known as an NWAY):

Table A1.10 Statements and Options Used to Load Cubes from Fully Summarized Data

Use these statements	Use these options	
PROC OLAP	CUBE=	Required
	PATH=	Required
	DESC=	Optional
METASVR	OLAP_SCHEMA=	Required
	REPOSITORY=	Optional
	HOST=	Optional
	PORT=	Optional
	PROTOCOL=	Optional
	USERID=	Optional
	PW=	Optional
DIMENSION	HIERARCHIES=	Required
	DESC=	Optional
	CAPTION=	Optional
	TYPE=TIME	Required only for TIME dimensions

Use these statements	Use these options	
	SORT_ORDER=	Optional
LEVEL		The LEVEL statement is optional unless you want to specify time periods for each level in a TIME dimension. If you specify a time period for one level, then you must specify a time period for all levels. To specify a time period, you use the TYPE= option.
HIERARCHY	LEVELS=	Required
	DESC=	Optional
	CAPTION=	Optional
MEASURE	STAT=	Required
	COLUMN | ANALYSIS=	Required
	AGGR_COLUMN=	Required
	DESC=	Optional
	CAPTION=	Optional
	UNITS=	Optional
	FORMAT=	Optional
	DEFAULT=	Optional
AGGREGATION	names of the contiguous levels to be used to create the aggregation	Required (additional AGGREGATION statements without the /TABLE= option can be used to create aggregations other than the automatically defined NWAY)
	/ TABLE=	Required

Maintaining Cubes

Building a Cube from an Existing Definition

It is possible to have cube definitions in the SAS Metadata Repository that do not have associated physical cubes. For example, you can use the DELETE_PHYSICAL= option in the PROC OLAP statement to delete a cube but leave its definition intact. You can also use SAS OLAP Cube Studio to save only the definition of a new cube.

The following table lists the PROC OLAP statements and options that you use to build a cube from an existing metadata definition:

Table A1.11 Statements and Options Used to Build a Cube from an Existing Definition

Use these statements	Use these options
PROC OLAP	CUBE=
METASVR	OLAP_SCHEMA=

Adding Aggregations to an Existing Cube

The following table lists the PROC OLAP statements and options that you use to add aggregations to an existing cube.

Table A1.12 Statements and Options Used to Add Aggregations to an Existing Cube

Use these statements	Use these options	
PROC OLAP	CUBE=	Required
METASVR	OLAP_SCHEMA=	Required
AGGREGATION	Names of the contiguous levels to be used to create the aggregation	Required
	/NAME=	Optional
	DATAPATH=	Optional
	INDEXPATH=	Optional
	COMPRESS | NOCOMPRESS	Optional
	INDEX | NOINDEX	Optional
	PARTSIZE=	Optional
	SEGSIZE=	Optional

Note: You can add and delete aggregations in the same PROC OLAP script. △

Note: You cannot add aggregations to a cube that contains aggregated data from a source other than the input data source. △

Deleting Aggregations from an Existing Cube

The following table lists the PROC OLAP statements and options that you use to delete aggregations from an existing cube.

Table A1.13 Statements and Options Used to Drop Aggregations from an Existing Cube

Use these statements	Use these options
DROP_AGGREGATION	Specify one or more level names that correspond to the aggregations that you want to remove, or use the aggregation name to specify the aggregation that you want to remove.
METASVR	OLAP_SCHEMA=
PROC OLAP	CUBE=

Note: You can add and delete aggregations in the same PROC OLAP script. △

Note: You cannot delete aggregations from a cube that contains aggregated data from a source other than the input data source. △

Deleting Cubes

The following table lists the PROC OLAP statements and options that you use to delete aggregations from an existing cube.

If you use the DELETE option, then both the physical cube and its definition, which is stored in the metadata server, are deleted.

Table A1.14 Statements and Options Used to Delete a Cube and Its Metadata

Use these statements	Use these options
METASVR	OLAP SCHEMA=
PROC OLAP	CUBE=
	DELETE

If you use the DELETE_PHYSICAL option, then only the physical cube is deleted; the definition remains intact.

Table A1.15 Statements and Options Used to Delete a Cube but Retain Its Metadata

Use these statements	Use these options
METASVR	OLAP_SCHEMA=
PROC OLAP	CUBE=
	DELETE_PHYSICAL

Specialized Syntax Options for PROC OLAP

Syntax Options for Managing Ragged Hierarchies

If a hierarchy is balanced, then all of its branches descend to the same level, and each member has a parent level that is positioned immediately above it. However, hierarchies are not always balanced and sometimes they contain missing hierarchy

members. To manage missing hierarchy members, you can use these four options, which were created specifically for ragged hierarchies:

Table A1.16 Options That Can Be Set to Manage Missing Hierarchy Members in Ragged Hierarchies

These options	Are available in these statements
EMPTY_CHAR=	PROC OLAP and HIERARCHY
EMPTY_NUM=	PROC OLAP and HIERARCHY
EMPTY=	LEVEL
IGNORE_EMPTY	HIERARCHY and LEVEL

Syntax Options Used for Performance

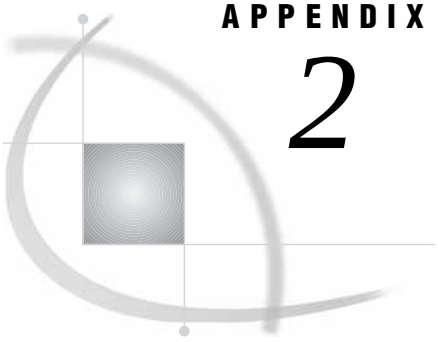
When you create a cube, you can set some options that can be used to optimize cube creation and query performance. If you set the options in the PROC OLAP statement, then the settings are applied to all aggregations in the cube. If you set the options in the AGGREGATION statement, then the options apply to that specific aggregation. Options set for individual aggregations override any options set in the PROC OLAP statement.

The options are

- INDEXSORTSIZE=
- MAXTHREADS=
- CONCURRENT=
- DATAPATH=
- INDEXPATH=
- COMPRESS | NOCOMPRESS
- NOINDEX | INDEX
- PARTSIZE=
- SEGSIZE= .

Note: INDEXSORTSIZE=, MAXTHREADS=, and CONCURRENT= are available only on the PROC OLAP statement. △

For an explanation of these options, see the PROC OLAP statement and the AGGREGATION statement.



APPENDIX

2

Recommended Reading

Recommended Reading

Here is the recommended reading list for this title:

- *Administrator for Enterprise Clients: User's Guide*
- *SAS Data Providers: ADO/OLE DB Cookbook*
- *SAS Language Reference: Concepts*
- *SAS Language Reference: Dictionary*
- *SAS Management Console: User's Guide*
- *SAS Metadata Server: Setup Guide*
- *SAS Open Metadata Architecture Reference*
- *SAS OLAP Server: Concepts and Excerpts from "MDX Solutions with Microsoft SQL Server Analysis Services"*
- *SAS OLAP Server: MDX Guide*
- SAS Companion that is specific to your operating environment

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

aggregation
a summary of detail data that is stored with or referred to by a cube. Aggregations support rapid and efficient answers to business questions.

aggregation table
a table that contains pre-calculated totals. Aggregation tables can be referred to by cubes, reducing the amount of time that is required for building the cubes.

ancestor
within a dimension hierarchy, a member that resides at a higher level in relation to other members in the hierarchy. For example, if a Geography dimension includes the levels Continent, Country, and City, then Europe and France would be ancestors of Paris , and Asia and Thailand would be ancestors of Bangkok.

ARM (Application Response Measurement)
an application programming interface that was developed by an industry partnership and which is used to monitor the availability and performance of software applications. ARM monitors the application tasks that are important to a particular business.

base table
a table that contains detail data that is used for building cubes or aggregation tables.

calculated member
in a dimension, a member whose value is derived from the values of other members.

cell
in a cube, the intersection that is defined by selecting one member from each dimension of that cube.

child
within a dimension hierarchy, a descendant in level n-1 of a member that is at level n. For example, if a Geography dimension includes the levels Country and City, then Bangkok would be a child of Thailand, and Hamburg would be a child of Germany.

cleanse
to improve the consistency and accuracy of data by standardizing it, reorganizing it, and eliminating redundancy.

cube
a logical set of data that is organized and structured in a hierarchical, multidimensional arrangement. A cube is a directory structure, not a single file. A cube includes measures, and it can have numerous dimensions and levels of data.

data cleansing
the process of eliminating inaccuracies, irregularities, and discrepancies from data.

data scrubbing
another term for data cleansing. See data cleansing.

descendant
in a dimension hierarchy, a member that resides at a lower level in relation to other members in the hierarchy. For example, if a Geography dimension includes the levels Country, State, and City, then California and Los Angeles would be descendants of USA .

detail data
nonsummarized (or partially summarized) factual information that pertains to a single area of interest, such as sales figures, inventory data, or human-resource data.

dimension
a group of closely related hierarchies. Hierarchies within a dimension typically represent different groupings of information that pertains to a single concept. For example, a Time dimension might consist of two hierarchies: (1) Year, Month, Date, and (2) Year, Week, Day. See also hierarchy.

dimension table
in a star schema, a table that contains the data for one of the dimensions. The dimension table is connected to the star schema's fact table by a primary key. The dimension table contains fields for each level of each hierarchy that is included in the dim ension.

drill down
in a view of an OLAP cube, to start at one level of a dimension hierarchy and to click through one or more lower levels until you reach the data that you are interested in.

drill up
in a view of an OLAP cube, to start at one level of a dimension hierarchy and to click through one or more higher levels until you reach the level of summarized data that you are interested in.

fact
a single piece of factual information in a data table. For example, a fact can be an employee name, a customer's phone number, or a sales amount. It can also be a derived value such as the percentage by which total revenues increased or decreased from one year to the next.

fact table
the central table in a star schema. The fact table contains the individual facts that are being stored in the database as well as the keys that connect each particular fact to the appropriate value in each dimension.

foreign key
a column or combination of columns in one table that references the corresponding primary key in another table. A foreign key must have the same attributes as the primary key that it references.

granularity
the relative level of detail that a data item represents. From the top of a dimension to the bottom, granularity increases. For example, in a Time dimension that consists

of a Year-Month-Day hierarchy, Month is more granular than Year, and Day is more gra nular than Month.

hierarchy
an arrangement of members of a dimension into levels that are based on parent-child relationships. Members of a hierarchy are arranged from more general to more specific. For example, in a Time dimension, a hierarchy might consist of the members Year, Qua rter, Month, and Day. In a Geography dimension, a hierarchy might consist of the members Country, State or Province, and City. More than one hierarchy can be defined for a dimension. Each hierarchy provides a navigational path that enables users to drill down to increasing levels of detail. See also member, level.

HOLAP (hybrid online analytical processing)
a type of OLAP in which relational OLAP (ROLAP) and multidimensional OLAP (MOLAP) are combined. In HOLAP, the source data is usually stored using a ROLAP strategy, and aggregations are stored using a MOLAP strategy. This combination usually results in the smallest amount of storage space. In HOLAP, aggregates can be pre-calculated and can be linked into a hybrid storage model.

leaf member
the lowest-level member of a hierarchy. Leaf members do not have any child members.

level
an element of a dimension hierarchy. Levels describe the dimension from the highest (most summarized) level to the lowest (most detailed) level. For example, possible levels for a Geography dimension are Country, Region, State or Province, and City.

MDDB (multidimensional database)
another term for cube. See cube.

MDX (multidimensional expressions) language
a standardized, high-level language that is used for querying multidimensional data sources. The MDX language is the multidimensional equivalent of SQL (Structured Query Language).

measure
a special dimension that contains summarized numeric data values that are analyzed. Total Sales and Average Revenue are examples of measures. For example, you might drill down within the Clothing hierarchy of the Product dimension to see the value of the Total Sales measure for the Shirts member.

member
a name that represents a particular data item within a dimension. For example, September 1996 might be a member of the Time dimension. A member can be either unique or non-unique. For example, 1997 and 1998 represent unique members in the Year level of a Time dimension. January represents non-unique members in the Month level, because there can be more than one January in the Time dimension if the Time dimension contains data for more than one year.

metadata profile
a definition of where a metadata server is located. The definition includes a host name, a port number, and a list of one or more metadata repositories. In addition, the metadata profile can contain a user's login information and instructions for connecti ng to the metadata server automatically.

metadata repository
a collection of related metadata objects, such as the metadata for a set of tables and columns that are maintained by an application. A SAS Metadata Repository is an example.

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

MOLAP (multidimensional online analytical processing)
a type of OLAP that stores aggregates in multidimensional database structures.

navigate
to purposefully move from one view of the data in a table (or in some other data structure, such as a cube) to another. Drilling down and drilling up are two examples of navigation.

NWAY aggregation
the aggregation that has the minimum set of dimension levels that is required for answering any business question. The NWAY aggregation is the aggregation that has the finest granularity. See also granularity.

OLE (Object Linking and Embedding)
a method of interprocess communication supported by Windows that involves a client/server architecture. OLE enables an object that was created by one application to be embedded in or linked to another application.

OLE DB
an open specification that has been developed by Microsoft for accessing both relational and nonrelational data. OLE DB interfaces can provide much of the same functionality that is provided by database management systems. OLE DB evolved from the Open Database Connectivity (ODBC) application programming interface. See also OLE (Object Linking and Embedding).

OLE DB for OLAP
an OLAP API that is used to link OLAP clients and servers by means of a multidimensional expressions (MDX) language. See also MDX (multidimensional expressions) language.

Open Metadata Architecture
See SAS Open Metadata Architecture.

parallel processing
a method of processing that divides a large job into several smaller jobs that can be executed in parallel on multiple CPUs.

parent
within a dimension hierarchy, the ancestor in level n of a member in level n-1. For example, if a Geography dimension includes the levels Country and City, then Thailand would be the parent of Bangkok, and Germany would be the parent of Hamburg. The paren t value is usually a consolidation of all of its children's values.

primary key
a column or combination of columns that uniquely identifies a row in a table.

reach-through
the act of retrieving and displaying to a user the (unsummarized) detail data from which the summarized data in a multidimensional database is derived, when that detail data is stored in a separate data repository.

result set
the set of rows or records that a server or other application returns in response to a query.

ROLAP (relational online analytical processing)
a type of OLAP in which the multidimensional data is stored in a relational database.

roll up
to summarize (or apply some other type of calculation or formula to) data values at one level of a dimension hierarchy in order to derive values for a parent level. For example, sales figures for January can be rolled up to Quarter1, and employee data for one department can be rolled up to the division level.

SAS ARM interface
an interface that can be used to monitor the performance of SAS applications. In the SAS ARM interface, the ARM API is implemented as an ARM agent. In addition, SAS supplies ARM macros, which generate calls to the ARM API function calls, and ARM system op tions, which enable you to manage the ARM environment and to log internal SAS processing transactions. See also ARM (Application Response Measurement).

SAS Metadata Repository
a repository that is used by the SAS Metadata Server to store and retrieve metadata. See also SAS Metadata Server.

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

SAS name
a name that is assigned to items such as SAS variables and SAS data sets. The first character must be a letter or an underscore. Subsequent characters can be letters, numbers, or underscores. Blanks and special characters (except the underscore) are not a llowed. The maximum length of a SAS name depends on the language element that it is assigned to. Many SAS names, such as names of DATA step variables and array names, can be 32 characters long. Others, such as librefs and filerefs, have a maximum length o f 8 characters.

SAS OLAP Cube Studio
a Java interface for defining and building OLAP cubes in SAS System 9 or later. Its main feature is the Cube Designer wizard, which guides you through the process of registering and creating cubes.

SAS OLAP Server
a SAS server that provides access to multidimensional data. The data is queried using the multidimensional expressions (MDX) language.

SAS Open Metadata Architecture
a general-purpose metadata management facility that provides metadata services to SAS applications. The SAS Open Metadata Architecture enables applications to exchange metadata, which makes it easier for these applications to work together.

SAS Open
Metadata Interface
a set of methods that enable users to read metadata from or write metadata to the SAS Metadata Server.

schema
a map or model of the overall data structure of a database. An OLAP schema specifies which group of cubes an OLAP server can access.

scrubbing
another term for data cleansing. See data cleansing.

shared dimension
a dimension that is used by more than one cube.

slice
a subset of data from a cube, where the data in the slice pertains to one or more members of one or more dimensions. For example, from a cube that contains data about customer feedback, one slice might pertain to feedback on one particular product (one me mber of the Product dimension). Another slice might pertain to feedback on that product from customers residing in particular geographic areas who submitted their feedback during a certain time period (one member of the Product dimension, multiple members of the Geography dimension, one or more members of the Time dimension).

sparsity
See data sparsity.

SPDE (Scalable Performance Data Engine)
a SAS engine that is able to deliver data to applications rapidly because it organizes the data into a streamlined file format. SPDE divides a problem (such as a WHERE clause) into smaller problems that can be processed in parallel. See also parallel proc essing.

SQL (Structured Query Language)
a standardized, high-level query language that is used in relational database management systems to create and manipulate database management system objects.

stacking
the act or process of storing individual aggregations in separate files. SAS OLAP Server supports stacking by storing generated MOLAP aggregations in individual files and allowing each aggregation to point to an external file.

star schema
tables in a database in which a single fact table is connected to multiple dimension tables. This is visually represented in a star pattern. SAS OLAP cubes can be created from a star schema.

stored statistics
statistics that are stored in a cube. Stored statistics can be used to derive higher-level statistics. Examples include sum, minimum, and maximum.

thread
a single path of execution of a process in a single CPU, or a basic unit of program execution in a thread-enabled operating environment. In a symmetric multiprocessing (SMP) environment, which uses multiple CPUs, multiple threads can be spawned and proces sed simultaneously. Regardless of whether there is one CPU or many, each thread is an independent flow of control that is scheduled by the operating system. See also threading, thread-enabled operating environment, SMP (symmetric multiprocessing).

threaded I/O
I/O that is performed by multiple threads in order to increase its speed. In order for threaded I/O to improve performance significantly, the application that is performing the I/O must be capable of processing the data rapidly as well. See also thread.

threading
a high-performance method of data I/O or data processing in which the I/O or processing is divided into multiple threads that are executed in parallel. In the

³boss-workerµ model of threading, the same code for the I/O or calculation process is executed s imultaeously in separate threads on multiple CPUs. In the ³pipelineµ model, a process is divided into steps, which are then executed simultaneously in separate threads on multiple CPUs. See also SMP (symmetric multiprocessing), parallel processing, parall el I/O.

Time dimension
a dimension that divides time into levels such as Year, Quarter, Month, and Day.

tuple
a data object that contains two or more components. In OLAP, a tuple is a slice of data from a cube. It is a selection of members (or cells) across dimensions in a cube. It can also be viewed as a cross-section of member data in a cube. For example, ([tim e].[all time].[2003], [geography].[all geography].[u.s.a.], [measures].[actualsum]) is a tuple that contains data from the Time, Geography, and Measures dimensions.

wizard
an interactive utility program that consists of a series of dialog boxes, windows, or pages. Users supply information in each dialog box, window, or page, and the wizard uses that information to perform a task.

Index

A

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

W

Your Turn

If you have comments or suggestions about *SAS 9.1 OLAP Server: Administrator's Guide*, please send them to us on a photocopy of this page, or send us electronic mail.

For comments about this book, please return the photocopy to

SAS Publishing
SAS Campus Drive
Cary, NC 27513
email: `yourturn@sas.com`

For suggestions about the software, please return the photocopy to

SAS Institute Inc.
Technical Support Division
SAS Campus Drive
Cary, NC 27513
email: `suggest@sas.com`