



## **SAS<sup>®</sup> 9 OLAP Server**

*Cube Building – Data Preparation and Cube Storage*

---

## Table of Contents

<b>Introduction .....</b>	<b>1</b>
<b>Overview of Multidimensional Databases .....</b>	<b>1</b>
Cube structure .....	1
Input data .....	2
Requirements for building cubes .....	3
Metadata requirements .....	3
Server requirements .....	3
<b>Input Data Preparation .....</b>	<b>4</b>
Format input data .....	4
Data input columns for measures .....	4
Format for input tables .....	4
Create additional categorical columns .....	6
Transform Parent/Child Tables .....	9
Prepare Ragged/Unbalanced Hierarchies .....	9
Resolve ambiguous data values .....	10
<b>Aggregation Design .....</b>	<b>10</b>
<b>Cube Storage Options .....</b>	<b>10</b>
Cube structure .....	11
MOLAP aggregation storage .....	11
ROLAP aggregation storage .....	11
Choosing between MOLAP and ROLAP aggregation storage .....	12
<b>Conclusion .....</b>	<b>13</b>

---

## Introduction

This technical paper introduces the terminology that is needed to understand multidimensional databases (cubes), provides an overview of the structure of a cube, and explains how to prepare data for loading into a multidimensional cube. This paper is written for database administrators and applications developers who build cubes. It is assumed that the reader has a basic understanding of relational databases and data warehousing concepts.

---

## Overview of Multidimensional Databases

### Cube structure

A multidimensional cube stores data in a way that enables fast data retrieval and, in particular, fast retrieval of summarized data. Data summarization in this context means condensing large numbers of detail records into meaningful numbers such as counts, sums, averages, or other statistical measures. For example, the activity in a checking account can be represented by either the list of all the transactions for a given period of time or by a summary report with a few significant measures, such as the total number of transactions and the start and ending balances.

Technically, a **Measure** is the combination of a numeric input column with a roll-up rule or statistic. You can create more than one measure from one input column. For example, you can create the measures “Sum of Amount” and “Maximum of Amount” from the input column “Amount.” In addition, you can create **Calculated Measures**, which are formulas that are based on the values of other measures.

The values in the measures change according to the grouping or **Dimension** that is being applied. For example, you can apply a grouping by time, which counts the total number of transactions for a given month, a specific year, or for several years. Another example of a dimension is a grouping by bank branch. Obviously, multiple dimensions can be applied, for example, listing summary numbers for the bank branches in a given month.

Dimensions can consist of more than one **Level**. Each level represents a different grouping within the same dimension. For example, a “Time” dimension can include the levels “Years,” “Months,” and “Days,” or an “Organization” dimension of a bank’s customer service centers can include the levels “Branches,” “States” (NC, SC, GA and so on), and “Regions” (South East, North East).

Levels within a dimension are grouped into one or more drill-down **Hierarchies**. For example, the sequence “Years,” “Months,” and “Days” represents a natural progression of hierarchy levels, from a larger group of detailed data to smaller groups. In the “Organization” dimension of a bank’s customer service centers, creating a hierarchy of “Region,” “State,” and “Branch” enables you to drill down from the most highly summarized level to more detailed levels. A dimension can have multiple hierarchies to provide different sequences of groupings. For example, a “Time” dimension can have a “Fiscal Year” hierarchy and a “Calendar Year” hierarchy.

Each combination of values within a dimension is called a **Member**. Some examples of members are shown here.

```
[Time].[2003]
[Time].[2004].[January]
[Time].[2004].[February].[12th]
```

For each dimension, there is also the special member ALL which represents the total for all members, for example, [Time].[All Time].

Not all categorical data attributes need to become a member of a hierarchy level. Some grouping information is needed only as additional information for a member or for applying subsets to data. These attributes can be loaded into member **Properties**. Properties can be associated with any level in a hierarchy.

This structure of dimensions, levels, hierarchies, members, and properties enables users to easily select data subsets and summarization levels and to easily navigate when querying a multidimensional cube. MDX (Multidimensional eXpression) is the language that is used to query multidimensional cubes. MDX uses dimension and member names and functions to specify data slices, summarization levels, and value comparisons. Here is an example of a simple MDX query.

```
SELECT [Time].[All Time].[2004].children on columns,
       [Measures].members on rows
from bank_cube
```

For a detailed introduction to the MDX language, see [SAS OLAP Server: Concepts and Excerpts from "MDX Solutions with Microsoft SQL Server Analysis Services"](#) by George Spofford (2003).

## Input data

A cube is always loaded from data in relational tables. This data can be stored either in SAS tables or in external RDBMSs and can be accessed through a wide selection of SAS data engines, that includes the Base SAS engine, SPD Server, SPD Engine, and the SAS/ACCESS engines to external RDBMSs. SAS software enables the user to be independent of the *physical* storage format of the data.

Usually, the data from which cubes are loaded resides in a static data repository that is updated at regular intervals. This information repository can be called a **Data Warehouse**, an **Enterprise Store**, or other name. Often, the tables in this repository are a more or less de-normalized replication of a specific time slice of the data in the operational systems.

Cubes can be loaded from data that is contained in any of the following types of tables.

- A **Detail Table** holds the numerical input data for the measures and columns that have categorical data.
- A **Star Schema (Fact Table and Dimension Tables)** contains the same information as a Detail Table, but a Star Schema consists of several tables. The input columns for the measures are stored in a **Fact Table**. The input columns for dimension levels and properties are stored in one table for each dimension, and

they are called **Dimension Tables**. The data records of the Fact table and the Dimension tables are linked via primary and foreign keys.

- A **Drillthrough Table**, which is optional, enables users to access the detail data at query time. The data needs to be in the form of a Detail table.
- **Aggregation Tables** contain pre-summarized data for any combination of dimension levels. This main performance feature enables cubes to access summarized numbers quickly. Aggregated data can be created and automatically stored in MOLAP (Multidimensional OLAP) tables as the cube is built or manually stored in ROLAP (Relational OLAP) tables and linked into the cube. For a detailed discussion of ROLAP and MOLAP aggregation storage, see the section “Cube Storage Options.” The topic about which Aggregation tables should be stored is discussed in the section “Aggregation Design.”

## Requirements for building cubes

### Metadata requirements

When creating a cube, metadata about the cube is stored in a SAS Open Metadata Repository. Each component of the SAS 9 Intelligence Architecture is made available on a SAS Open Metadata Repository. This enables sharing of all application elements by both internal and external applications, including server definitions, cubes, tables, libraries, and the security that is applied to them.

In order to load cubes, a SAS Open Metadata Server must be running and should include the following elements.

- At least one repository.
- An OLAP schema.
- A database server and a database schema which is required only when using SAS/ACCESS tables.
- One or more libraries.
- One or more table data sources.
- A SAS Workspace Server and an Object Spawner (optional) when using the Cube Designer wizard. The SAS Workspace Server is not needed when using PROC OLAP.
- User-defined formats and informats (if they are being used in the input data columns).

### Server requirements

An OLAP server is NOT necessary to create a cube. You can use either PROC OLAP, which is part of Base SAS software, and run it as a batch job or as a stored process, or

the Cube Designer wizard, which is available in OLAP Cube Studio and SAS ETL Studio. In the latter case, a SAS Workspace Server and an Object Spawner (optional) is needed.

---

## Input Data Preparation

This section describes the expected format of your cube input data tables and columns. It also discusses considerations for data values when loading cubes.

### Format input data

The columns in relational data tables are either used as input columns for cube measures, or they map to a dimension level or property. You need to identify the columns that hold numerical data to be used as input for your measures, and make sure that your data tables have all the columns that map to dimension levels.

### Data input columns for measures

Measures are loaded from the data entities that you want to summarize from. One input column can load one or more measures. Not all measures are directly derived from input columns. Calculated measures are formulas that are based on the values of other measures. An example of a formula to calculate the percentage contribution of a value to its parent (such as the value on a given day with respect to the whole month) is shown here.

$$\frac{[\text{Measures}].[\text{Sum of Amount}]}{([\text{Measures}].[\text{Sum of Amount}], [\text{Time}].\text{CurrentMember.Parent})}$$

You don't need data input columns for calculated measures. You define calculated measures with the cube.

### Format for input tables

The input tables for loading a cube must be in the following formats.

#### Detail Table (also known as Basetable)

A Detail table includes all the columns that are needed to load the hierarchy levels, the property levels, and the measures. A Detail table needs the following elements.

- One column per dimension level (char or num)
- One column for each property (char or num)
- Numeric analysis variables.

**Star Schema**

A Fact table needs the following elements.

- One key column with a foreign key for each dimension (char or num)
- Numeric analysis variables.

A Dimension table needs the following elements.

- One key column with a primary key (char or num)
- One column for each dimension level (char or num)
- One column for each property (char or num).

**Note:** You cannot have more than one property value for each distinct value of the corresponding level.

To successfully load a cube, all foreign keys in the Fact table need to have a corresponding primary key in a Dimension Table.

The input columns for the levels and properties of some dimensions can be part of the Fact table instead of being in a separate, linked Dimension table.

**Summary Data Set (also known as N-way data set)**

You can load a cube from one data set that is already summarized. The Summary table needs the following elements.

- One column per dimension level (char or num)
- One column for each property (char or num)
- One column per measure, summarized by the appropriate statistic for the measure.

For additional information about stored statistics see the section “ROLAP Aggregation Storage.”

**Drillthrough Table**

Many OLAP applications give users the ability to select a cell or a range of cells and then view the input data that the cell data was summarized from. Specifying a Drillthrough table for your cube provides that functionality. The Drillthrough table must have the same columns with the same attributes (name, type, format) as the tables that the cube was loaded from. In many cases, the Detail table is also the Drillthrough table. For a Star Schema, a view that fully joins the Star Schema is the Drillthrough table. When loading from a Summary table, that Summary table can be used as a Drillthrough table.

**Aggregation Tables**

In addition to a Summary table for the N-way aggregation, Summary tables can also be provided for other aggregations. You cannot load the cube from those data sets — a full

set of data is needed to load a cube, and any aggregation other than the N-way misses at least one dimension level. Aggregation tables need the following elements.

- One column for each dimension level in the aggregation
- One column per measure.

The column names and attributes (type, format) must be consistent across all input tables for a cube. Also, it is assumed that the data is consistent. For example, if you provide Aggregation tables, the cube assumes that those tables are summarizations from the data that is provided to load the cube. If the data is out of synch, then the numeric results that are returned to queries against the cube are unpredictable.

### Create additional categorical columns

Dimension levels and properties are loaded from one input column each. In some instances, not all dimension levels are represented as columns. For example, your data might only contain a column for Date, but you also want Months and Years as levels in your cube. In another example, your data might contain only customer IDs, but you want to group them by region or by the first letter of the last name. You will need to create those columns prior to loading the cube.

Figure 1 depicts the process of introducing additional levels into a customer-specific dimension. The **date** key in the TIMEDIM table is being formatted into columns that can be used as input for various Time dimension levels.

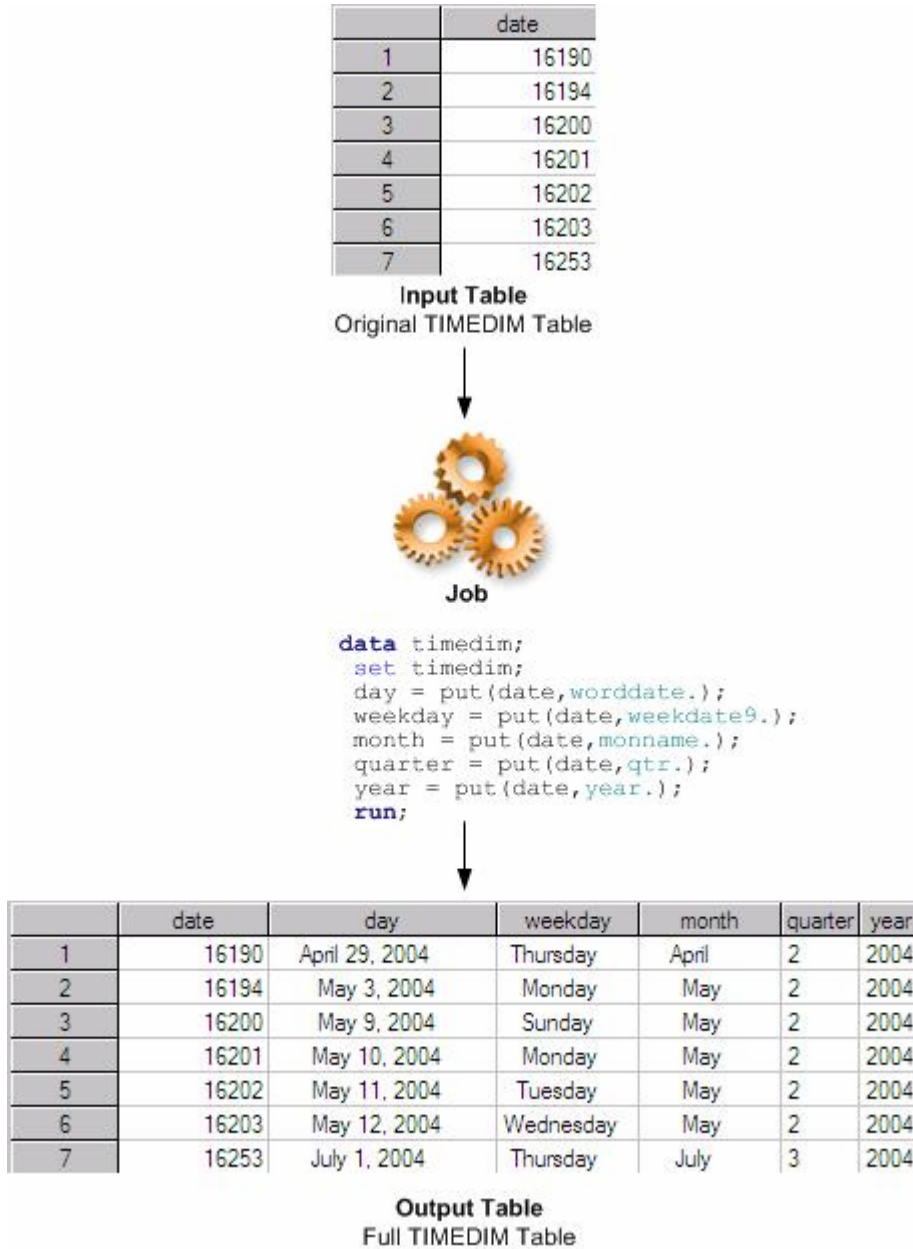


Figure 1. Introducing Additional Levels into a Dimension

If your data is normalized, and depending on the complexity of your database, you might have to replace some foreign keys with their attribute columns. In other words, you might have to perform table joins so that the data for the levels and properties of each dimension are available in one table. This is important because you need the data in the form of a Star Schema (or possibly a Detail table) in order to be able to load a cube.

Figure 2 depicts a sample table join. The “city” information in the CUSTOMER table is linked to a separate CITY table. Join the tables by resolving the **city** keys with the desired information to create an input dimension table for your cube.

	cust_id	cust_firstname	cust_lastname	cust_city_id
1	456	John	Smith	45
2	765	Ann	Taylor	45
3	234	Tara	Black	46

	city_id	city_name
1	45	Montreal
2	46	Toronto

**Input Table**  
CUSTOMER Table

**Input Table**  
CITY Table



```
proc sql;
  create table custdim as
  select cust.cust_id,
         cust.cust_firstname,
         cust.cust_lastname,
         city.city_name
  from customer cust,
       city city
  where cust.cust_city_id = city.city_id;
quit;
```

	date	day	weekday	month	quarter	year
1	16190	April 29, 2004	Thursday	April	2	2004
2	16194	May 3, 2004	Monday	May	2	2004
3	16200	May 9, 2004	Sunday	May	2	2004
4	16201	May 10, 2004	Monday	May	2	2004
5	16202	May 11, 2004	Tuesday	May	2	2004
6	16203	May 12, 2004	Wednesday	May	2	2004
7	16253	July 1, 2004	Thursday	July	3	2004

**Output Table**  
CUSTDIM Table

Figure 2. Sample Table Join

## Transform Parent/Child Tables

Sometimes the hierarchical relationship of your dimension is stored in a Parent/Child table where one member of the relationship is stored in a parent column and the other is stored in a child column. In order to be able to load a cube dimension, Parent/Child tables need to be transformed into tables that have one column per level.

The following example shows a simple Parent/Child table (Table 1) and the transformed, leveled Dimension table (Table 2).

	child	parent	empid
1	John Gray	Isabelle Green	123
2	Isabelle Green	Alfred Black	124
3	Bart Blue	Isabelle Green	125
4	Donald Red	Alfred Black	126

Table 1. Parent-Child Table

	level1	level2	level3	empid
1	Alfred Black	Isabelle Green	John Gray	123
2	Alfred Black	Isabelle Green	Bart Blue	125
3	Alfred Black	Isabelle Green		124
4	Alfred Black	Donald Red		126

Table 2. Leveled Dimension Table

## Prepare Ragged/Unbalanced Hierarchies

In some cases, not all positions in a leveled hierarchy are taken. For example, as shown in Figure 3, in a geographical dimension, some cities might contribute directly to the Region total, while others are grouped into States.

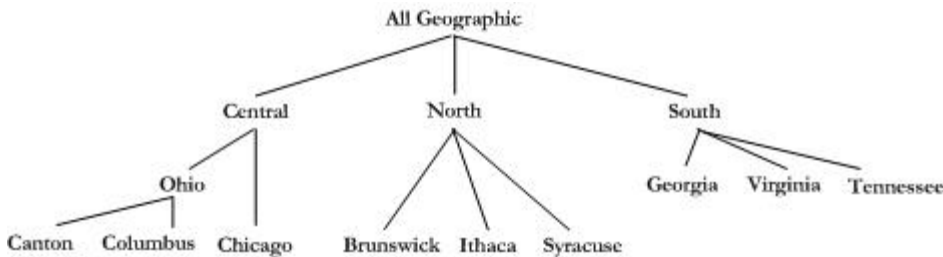


Figure 3. Geographic Hierarchy

The missing values at certain levels can be marked by special values in the input dimension columns. The default value for indicating these “holes” (see Table 3) is a blank or missing value for character columns and a dot ( . ) or null value (blank) for numeric columns.

	level1	level2	level3	empid
1	Alfred Black	Isabelle Green	John Gray	123
2	Alfred Black	Isabelle Green	Bart Blue	125
3	Alfred Black	Isabelle Green		124
4	Alfred Black		Donald Red	126

**Table 3. Marking Missing Values**

## Resolve ambiguous data values

The MDX query language is not case-sensitive, and it ignores leading blanks. For example, the members [New York], [NEW YORK], and [ New York] (notice the leading blank) are considered identical.

If ambiguous input data values are encountered during cube load, then the cube cannot be successfully loaded. You can use utilities such as the [SAS function PROPCASE](#) or [SAS® Data Quality Server software](#) to resolve ambiguities.

---

## Aggregation Design

The main way in which a multidimensional cube is able to deliver fast query response times is by storing summarized data. Any combination of dimension levels — as long as it is appropriate within the defined hierarchies — can become a stored aggregation. Which aggregations are being stored has a direct effect on the SAS OLAP Server CPU usage, file I/O, and query response times. The aggregations that are being stored also affect cube build time and absolute cube file size. Therefore, it's a trade-off between one-time resource use at cube build time and many-time resource use at cube query time.

In choosing the best aggregations to summarize and store with the cube, the most important factor to take into account is user behavior. It is recommended that you start with an initial aggregation design that is based on a minimal set of aggregations or on your best assumptions about usage patterns. After the cube is deployed to users, you can use the ARM (Application Response Management) logging capabilities of the SAS OLAP Server to collect data about the usage pattern and the performance of individual queries. You can analyze the collected data to find out which cube aggregations are used most, which aggregations you can safely eliminate without harming query performance, and which aggregations are often requested but don't exist in the cube. The ARM data provides all the information that you need in order to get the optimal query improvement for your build time and cube storage space.

---

## Cube Storage Options

This section explains which options are available for storing aggregated data with your SAS OLAP cube.

## Cube structure

A SAS OLAP cube consists minimally of a number of files that contain information about the structure of the cube and the values of the dimension members. A cube always has these files. Aggregated data values can also be stored with the cube (optional).

The aggregated data values for SAS OLAP cubes can be stored either with the cube in the cube's internal format or external to the cube in relational summary tables. MOLAP aggregation storage is the cube-internal storage for aggregations. ROLAP aggregation storage is the cube-external summary tables.

MOLAP aggregation tables are created as part of the cube creation step. ROLAP summary tables need to be pre-calculated from the input data (using tools such as SQL or PROC MEANS/PROC SUMMARY) and made known to the cube at cube creation time.

## MOLAP aggregation storage

The MOLAP storage of SAS OLAP cubes stores the cube data in the same table format as the format that is used by the SAS SPD Engine. The MOLAP storage takes advantage of key contraction and allows data access by using the cube's internal data representation directly. In other words, for its internal processing, the SAS OLAP Server represents each hierarchy member (for example, [1999].[January].[01]) as a numeric key. The MOLAP storage uses that internal key. This reduces the number of columns that are needed to identify dimension levels and enables direct communication between the SAS OLAP Server and the physical storage, with no need for data conversion and look up.

The MOLAP storage of SAS OLAP cubes has the same threading and scalability features as the files used by the SAS SPD Engine. The data and the index section of the files are stored in different physical files. This enables parallel access to the data and index sections. The data and index files themselves are stored in partitions, enabling parallel data retrieval within the same file. The partitions of the data and the index section can be distributed over multiple disc controllers, thus adding a further boost to the threaded and partitioned architecture by reducing contention and possible bottlenecks in the physical I/O.

## ROLAP aggregation storage

ROLAP tables used in SAS OLAP cubes can be SAS data sets, SAS data views, or any tables or views accessible via a SAS engine. That extends the choice of available storage options for SAS OLAP cubes to include SPDE, SPDS, and any RDBMS product for which a SAS/ACCESS software product is available.

ROLAP aggregation tables need to conform to the structure of the input data. The columns that feed the dimension levels need to have the same column names and attributes that were used in the input data when loading the cube. All aggregations need to be stored in fully de-normalized form.

Use the following rules to make aggregation columns for measures available.

- Each ROLAP aggregation table must include all columns for the cube's measures with stored statistics.

- SAS OLAP cube aggregations only store the following six summarizable statistics: SUM, N, NMISS, USS, MIN, and MAX. The other available statistics are derived from the stored statistics by internal calculations. For example, in order to include a measure for the AVG statistic in your cube, you need to make columns available in your ROLAP aggregation tables that were generated by using SUM and N (count).

Table 4 shows which stored statistics must be available for specific derived statistics:

Derived Statistics	Required Stored Statistics
AVG	N, SUM
CSS	N, SUM, USS
RANGE	MIN, MAX
VAR, STD, STDERR, CV, T, PRT, LCLM, UCLM	N, SUM, USS

**Table 4. Requirements for Derived and Stored Statistics**

Data requests against ROLAP tables that are part of a cube are sent as simple SQL statements for execution and aggregation to the respective database engine.

ROLAP data requests can also run against the data that was used to create and load the cube, whether from a Detail table or a Star Schema. The cube's input data can be used in place of the aggregation with the combination of the lowest level of all dimensions (often called NWAY or NWAY aggregation, which is a name borrowed from PROC MEANS/PROC SUMMARY where it denotes the combination of all CLASS variables).

### Choosing between MOLAP and ROLAP aggregation storage

The SAS OLAP cube MOLAP store is optimized for SAS OLAP Server internal processing; has a minimal data-size footprint; uses threaded, parallel data access; and is tunable to any hardware environment. It is convenient because it doesn't require additional data management steps.

ROLAP aggregation storage enables you to use existing ROLAP schemas and reuse legacy SAS OLAP Server SAS 8 HOLAP structures. ROLAP aggregation storage enables users to use database systems and data servers of their choice to store and serve cube aggregation data. Existing processes can be used to create and access aggregation data to offload and distribute data access, I/O, and rollup to server systems of the user's choice.

A hybrid approach is possible. For example, users with existing ROLAP structures can build a "light" SAS OLAP cube with no additional stored aggregations and add MOLAP aggregations to further tune the cube performance.

---

## Conclusion

SAS 9 OLAP cubes are built from relational input data. To build cubes successfully, the data in input files must be internally consistent, and the columns of input data sets need to reflect the dimensional levels of the cubes. When building an OLAP application, it is prudent to assume that building the actual cube is, by far, the smallest part of the project. The most effort has to be concentrated on finding and collecting the data, making the data consistent, and transforming the data to adhere to the dimensional design.







World Headquarters  
and SAS Americas  
SAS Campus Drive  
Cary, NC 27513 USA  
Tel: (919) 677 8000  
Fax: (919) 677 4444  
U.S. & Canada sales:  
(800) 727 0025

SAS International  
PO Box 10 53 40  
Neuenheimer Landstr. 28-30  
D-69043 Heidelberg, Germany  
Tel: (49) 6221 4160  
Fax: (49) 6221 474850  
**[www.sas.com](http://www.sas.com)**