SAS® Visual Forecasting 8.2 Forecasting Procedures

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2017. *SAS® Visual Forecasting 8.2: Forecasting Procedures*. Cary, NC: SAS Institute Inc.

SAS® Visual Forecasting 8.2: Forecasting Procedures

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

June 2018

Contents

Chapter 1. What's New in SAS Visual Forecasting 8.2 Procedures 1
Chapter 2. Introduction 3
Chapter 3. Shared Concepts 7
Chapter 4. The TSINFO Procedure 11
Chapter 5. The TSMODEL Procedure 19
Chapter 6. The TSRECONCILE Procedure 61

Subject Index **77**

Syntax Index **79**

Chapter 1

What's New in SAS Visual Forecasting 8.2 Procedures

Contents

Overview **1**
New Procedure **1**
TSINFO Procedure 1
Procedures Enhancements **1**
TSMODEL Procedure Package Additions 1

Overview

SAS Visual Forecasting 8.2 offers one new procedure and supports additional time series packages for the TSMODEL procedure. The TSMODEL procedure is also provided in SAS Econometrics 8.2 procedures.

New Procedure

TSINFO Procedure

The new TSINFO procedure evaluates a variable in an input data table for its suitability as a time ID variable in SAS procedures and solutions that are used for time series analysis.

Procedures Enhancements

TSMODEL Procedure Package Additions

The TSMODEL procedure adds support for the following new packages:

- The singular spectrum analysis (SSA) package enables you to perform SSA forecasting.
- The time filter (TIMFIL) package enables you to filter time series vectors.
- The time series motif (MTF) package enables you to search for patterns in time series.

Chapter 2
Introduction

Contents

Overview of SAS Visual Forecasting Procedures **3**
About This Book **3**
Chapter Organization 4
Typographical Conventions 4
Options Used in Examples 5
Where to Turn for More Information **5**
Online Documentation 5
SAS Technical Support Services 5

Overview of SAS Visual Forecasting Procedures

SAS Visual Forecasting procedures provide time series analysis and forecasting tools that have been specially developed to take advantage of the distributed environment that SAS Viya provides. Methods include time series analysis (time domain and frequency domain), time series decomposition and filtering, time series modeling, time series forecasting and monitoring, and temporal data mining (motif discovery and time series distance measures). The procedures provide time series diagnostics, automatic variable selection and calendar events, automatic time series model selection for forecasting and monitoring, and out-of-sample performance monitoring.

NOTE: When you license SAS Visual Forecasting, you also have access to SAS/ETS software and SAS Forecast Server Procedures. For more information about SAS/ETS procedures, see *SAS/ETS User's Guide*. For more information about SAS Forecast Server Procedures, see *SAS Forecast Server Procedures: User's Guide*.

About This Book

This book assumes that you are familiar with Base SAS software and with the books *SAS Language Reference: Concepts* and *SAS Visual Data Management and Utility Procedures Guide*. It also assumes that you are familiar with basic SAS System concepts, such as using the DATA step to create SAS data sets and using Base SAS procedures (such as the PRINT and SORT procedures) to manipulate SAS data sets.

Chapter Organization

This book contains the following chapters:

- Chapter 2, this chapter, provides an overview of SAS Econometrics procedures and summarizes related information, products, and services.
- Chapter 3 describes how to use SAS Cloud Analytic Services (CAS) sessions and how to load a SAS data set onto a CAS server.
- The remaining chapters describe the procedures; they appear in alphabetical order by procedure name and are organized as follows:
  - The "Overview" section briefly describes the analysis provided by the procedure.
  - The "Getting Started" section provides a quick introduction to the procedure through a simple example.
  - The "Syntax" section describes the SAS statements and options that control the procedure.
  - The "Details" section discusses methodology and other topics, such as ODS tables.
  - The "Examples" section contains examples that use the procedure.
  - The "References" section contains references for the methodology.

Typographical Conventions

This book uses several type styles for presenting information. The following list explains the meaning of the typographical conventions used in this book:

roman	is the standard type style used for most text.
UPPERCASE ROMAN	is used for SAS statements, options, and other SAS language elements when they appear in text. However, you can enter these elements in your own SAS programs in lowercase, uppercase, or a mixture of the two.
UPPERCASE BOLD	is used in the "Syntax" sections' initial lists of SAS statements and options.
oblique	is used in the syntax definitions and in text to represent arguments for which you supply a value.
VariableName	is used for the names of variables and data sets when they appear in text.
bold	is used for matrices and vectors.
italic	is used for terms that are defined in text, for emphasis, and for references to publications.
`monospace`	is used for example code. In most cases, this book uses lowercase type for SAS code.

Options Used in Examples

The HTMLBLUE style is used to create the graphs and the HTML tables that appear in the online documentation. The PEARLJ style is used to create the PDF tables that appear in the documentation. A style template controls stylistic elements such as colors, fonts, and presentation attributes. You can specify a style template in an ODS destination statement as follows:

```
ods html style=HTMLBlue;
. . .
ods html close;

ods pdf style=PearlJ;
. . .
ods pdf close;
```

Most of the PDF tables are produced by using the following SAS System option:

```
options papersize=(6.5in 9in);
```

If you run the examples, you might get slightly different output. This is a function of the SAS System options that are used and the precision that your computer uses for floating-point calculations.

Where to Turn for More Information

Online Documentation

You can access the documentation by going to `http://support.sas.com/documentation`.

SAS Technical Support Services

The SAS Technical Support staff is available to respond to problems and answer technical questions regarding the use of procedures in this book. Go to `http://support.sas.com/techsup` for more information.

Chapter 3
Shared Concepts

Contents

Introduction to Shared Concepts **7**
Using CAS Sessions and CAS Engine Librefs 7
Loading a SAS Data Set onto a CAS Server 8

Introduction to Shared Concepts

SAS Visual Forecasting procedures run on SAS Viya. One component of SAS Viya is SAS Cloud Analytic Services (CAS), which is the analytic server and associated cloud services. The following subsections describe how to set up and use CAS sessions.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named mysess, and the LIBNAME statement creates the mycas CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the mysess session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS statement and the LIBNAME statement, see *SAS Cloud Analytic Services: Language Reference*. For general information about CAS and CAS sessions, see *SAS Cloud Analytic Services: Fundamentals*.

Loading a SAS Data Set onto a CAS Server

Procedures in this book require the input data to reside on a CAS server. To work with a SAS data set, you must first load the data set onto the CAS server. Data loaded on the CAS server are called *data tables*. This section lists three methods of loading a SAS data set onto a CAS server. In this section, mycas is the name of the caslib that is connected to the mysess CAS session.

- You can use a single DATA step to create a data table on the CAS server as follows:

  ```
  data mycas.Sample;
     input y x @@;
     datalines;
  .46 1 .47 2 .57 3 .61 4 .62 5 .68 6 .69 7
  ;
  ```

  Note that DATA step operations might not work as intended when you perform them on the CAS server instead of the SAS client.

- You can create a SAS data set first, and when it contains exactly what you want, you can use another DATA step to load it onto the CAS server as follows:

  ```
  data Sample;
     input y x @@;
     datalines;
  .46 1 .47 2 .57 3 .61 4 .62 5 .68 6 .69 7 .78 8
  ;
  data mycas.Sample;
     set Sample;
  run;
  ```

- You can use the CASUTIL procedure as follows:

  ```
  proc casutil sessref=mysess;
     load data=Sample casout="Sample";
  quit;
  ```

  The CASUTIL procedure can load data onto a CAS server more efficiently than the DATA step. For more information about the CASUTIL procedure, see *SAS Cloud Analytic Services: Language Reference*.

The mycas caslib stores the Sample data table, which can be distributed across many machine nodes. You must use a caslib reference in procedures in this book to enable the SAS client machine to communicate with the CAS session. For example, the following TSMODEL procedure statements use a data table that resides in the mycas caslib:

```
proc tsmodel data = mycas.Sample;
  ...statements...;
run;
```

You can delete your data table by using the DELETE procedure as follows:

```
proc delete data = mycas.Sample;
run;
```

The Sample data table is accessible only in the mysess session. When you terminate the mysess session, the Sample data table is no longer accessible from the CAS server. If you want your Sample data table to be available to other CAS sessions, then you must *promote* your data table. For more information about data tables, see *SAS Cloud Analytic Services: User's Guide*.

Chapter 4
The TSINFO Procedure

Contents

Overview: TSINFO Procedure **11**
PROC TSINFO Compared with the TIMEID Procedure 11
Getting Started: TSINFO Procedure **12**
Syntax: TSINFO Procedure **13**
Functional Summary 13
PROC TSINFO Statement 13
BY Statement 14
ID Statement 14
Details: TSINFO Procedure **15**
Time ID Diagnostics 15
Inferring Time Intervals and Alignments 15
Printed Tabular Output 16
Examples: TSINFO Procedure **17**
Example 4.1: Examining a Weekly Time ID Variable and Inferring a Date Interval . . 17
Example 4.2: Examining Multiple BY Groups 17

Overview: TSINFO Procedure

The TSINFO procedure evaluates a variable in an input data table for its suitability as a time ID variable in SAS procedures and solutions that are used for time series analysis. PROC TSINFO assesses how well a time interval specification fits SAS date values, SAS datetime values, or observation numbers that are used to index a time series. The time interval used in this analysis can be either specified explicitly as input or inferred based on values of the time ID variable. The TSINFO procedure produces diagnostic information in the form of data tables and ODS tables. These diagnostic results summarize characteristics of the time ID variable that can help determine its use as an index in other time series procedures and solutions.

PROC TSINFO is intended for use as a tool to either identify the time interval of a variable or prepare problematic data sets for use in subsequent time series analyses. In particular, this procedure can be used to investigate inconsistencies between time ID values and the ID statement options that are used in other SAS procedures and solutions.

PROC TSINFO Compared with the TIMEID Procedure

The TSINFO procedure is the next generation of the TIMEID procedure (a SAS/ETS procedure) for time series information analysis. PROC TSINFO was developed specifically for SAS Viya. The syntax of PROC

TSINFO is similar to the syntax of PROC TIMEID, from which it borrows the underlying methodology and goals. PROC TSINFO is designed to run on a cluster of machines that distribute the data and computations to multiple threads, and it requires your data tables to be available on a SAS Cloud Analytic Services (CAS) server.

Table 4.1 shows the differences between the TSINFO and TIMEID procedures.

Table 4.1 PROC TSINFO versus PROC TIMEID

Feature	TSINFO	TIMEID
Threading	Multithreaded	Single-threaded
Input data	CAS tables	SAS data sets
Requires sorted input data?	No	Yes, by BY variables and ID variable bottom-up

Getting Started: TSINFO Procedure

When a data table contains a time ID variable that has corrupted, missing, or duplicate values, PROC TSINFO can help isolate and identify these problematic observations. For larger data tables whose quality is unknown, it can be useful to get a general overview of the relative number of observations that have problematic time ID values. When prior knowledge of the time interval that separates observations is incomplete, PROC TSINFO can be used to infer the interval.

The following example uses the Sashelp.Air data set to illustrate how to analyze some time series information.

The following DATA step loads the Air data set from the Sashelp directory to a table named mycas.air in your CAS session. This DATA step assumes that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.air(replace=yes);
   set sashelp.air;
run;
```

The following statements use the TSINFO procedure to infer the interval of the time series:

```
proc tsinfo data=mycas.air
            nthreads=2
   ;
   id date align=BEGIN;
   by class;
run;
```

PROC TSINFO requires the following inputs:

- The input table, specified in the DATA= option in the PROC statement

- BY variables (class), specified in the BY statement
- The variable that contains the timestamps in the ID statement

Syntax: TSINFO Procedure

The following statements are available in the TSINFO procedure:

PROC TSINFO < *options* > **;**
  BY *variables* **;**
  ID *variable* < *options* > **;**

The PROC TSINFO and ID statements are required. The following sections describe the PROC TSINFO statement and then describe the other statements in alphabetical order.

Functional Summary

Table 4.2 summarizes the statements and options that are used with the TSINFO procedure.

Table 4.2 PROC TSINFO Functional Summary

Description	Statement	Option
Data Options		
Specifies the input data table	PROC TSINFO	DATA=
Specifies the detailed output interval data table	PROC TSINFO	OUTINTERVALDETAILS=
Specifies the number of threads to be used in the procedure	PROC TSINFO	NTHREADS=
Time ID Options		
Specifies the alignment of time ID values	ID	ALIGN=

PROC TSINFO Statement

PROC TSINFO *options* **;**

The PROC TSINFO statement invokes the TSINFO procedure.

You must specify the following data tables:

DATA=*CAS-libref.data-table*
specifies the SAS input data table that contains the input data for the procedure. If the DATA= option is not specified, the most recently created SAS data set is used. *CAS-libref.data-set* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table.

OUTINTERVALDETAILS =*CAS-libref.data-table*
names the output data table to contain the time ID interval information for each BY group in the DATA= data set. For more information, see the section "OUTINTERVALDETAILS= Data Set" on page 16. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table.

You can also specify the following *option*:

NTHREADS=*number*
specifies the number of threads to use in the computation.

By default, NTHREADS=1.

BY Statement

BY *variables* **;**

You can use a BY statement to obtain separate analyses for groups of observations that are defined by the *variables*, which can be either character or numeric.

When a BY statement is included, the procedure expects the input data to be sorted in the order of the BY variables.

ID Statement

ID *variable* < /*options* > **;**

The ID statement names a numeric *variable* that identifies observations in the input and output data. The ID variable values are assumed to be SAS date or datetime values.

You can specify the following *options* to determine how to space the time ID values and align them relative to a SAS date or datetime interval:

ALIGN=BEGINNING | MIDDLE | ENDING | INFER
specifies the alignment of the identifying SAS date or datetime that is used to represent intervals. The value of the ALIGN= option is used in the analysis of the time ID variable. You can specify the following values:

BEGINNING | BEG | B	uses the beginning date in the interval as the identifying date for the interval.
MIDDLE | MID | M	uses the middle date in the interval as the identifying date for the interval.
ENDING | END | E	uses the end date in the interval as the identifying date for the interval.

INFER infers that the alignment of values within time intervals from the time ID values.

By default, ALIGN=BEGIN.

Details: TSINFO Procedure

Time ID Diagnostics

For a specified time interval, PROC TSINFO decomposes the raw time ID values in an input data table into the following three quantities, whose values are represented by nonnegative integers at each unique time ID value in the input series:

intervalcounts the number of observations that share each time interval in the data set.

offsets the numerical difference between a time ID value and the aligned value for that time interval. The unit of measure that is used to express this distance is days for date values and seconds for datetime values. The offset is computed for each time ID value, t_i, by using the following SAS expression:

$$\text{offset}_i = t_i - \text{INTNX}(\text{interval}, t_i, 0, \text{alignment})$$

spans the number of intervals between each time ID value and the previous time ID value. The spans value is equivalent to the number returned by the following SAS expression:

$$\text{spans}_i = \text{INTCK}(\text{interval}, t_{i-1}, t_i)$$

Inferring Time Intervals and Alignments

A time interval is inferred from the time ID values in the input data. The technique that is used to infer a time interval involves searching for the interval that fits the greatest number of time ID values. First, time ID values are sampled from the input data to generate a set of candidate intervals. Then the candidate interval that is consistent with greatest number of time ID values is chosen to represent the time series.

When the ALIGN=INFER option is specified, the convention that is used to specify time interval alignment is inferred from the time ID variable values by using a similar technique. When both the time interval and its alignment are to be inferred, each of the possible alignments (BEGIN, MIDDLE, and END) is considered in the search. Precedence in the search is given to intervals that use the BEGIN alignment.

Data Table Output

The TSINFO procedure creates an ODS output data table. This data table contain the variables that are specified in the BY statement along with variables that characterize the time ID values. The information in this data table summarizes time ID diagnostic information across all BY groups in the table that is specified in the DATA= option.

OUTINTERVALDETAILS= Data Set

The OUTINTERVALDETAILS= data set contains statistics about the time interval that is specified in the ID statement or inferred from the time ID values for each BY group. The following variables represent these statistics:

TIMEID	time ID variable name
INTERVAL	time interval that was specified or is recommended
INTNAME	time interval base name that was specified or is recommended
MULTIPLIER	time interval multiplier that was specified or is recommended
SHIFT_INDEX	time interval shift index that was specified or is recommended
ALIGNMENT	time interval alignment that was specified or is recommended
SEASONALITY	seasonality determined from the specified or recommended time interval
FORMAT	format of the time ID variable

Printed Tabular Output

The TSINFO procedure produces printed output by using the Output Delivery System (ODS). By default, the procedure produces the overall time series information table.

Table 4.3 ODS Tables Produced in PROC TSINFO

ODS Name	Description
DataSet	Information about the input data table
Decomposition	Time ID counts, offsets, and spans
Interval	Information about the time interval
IntervalCountsComponent	Frequency distribution of interval counts
Align	Time ID alignment information
Seasonality	Summary of the number of valid observations

Examples: TSINFO Procedure

Example 4.1: Examining a Weekly Time ID Variable and Inferring a Date Interval

This example illustrates how a time ID variable can be inferred from data when a sufficient number of observations are present. The following DATA step loads the Air data set onto the CAS server. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.air(replace=yes);
   set sashelp.air;
run;
```

The following TSINFO procedure statements generate an ODS display of the time series that characterizes interval counts, alignment, and seasonality in the time ID variable:

```
proc tsinfo data=mycas.air outintervaldetails=mycas.interval2 nthreads=2;
  id date align=B;
run;
```

There are 144 observations in the mycas.Air data table. The 144 observations are enough to determine that the Date time ID variable is represented by the MONTH interval.

Example 4.2: Examining Multiple BY Groups

This example illustrates how a time ID variable can be examined independently over each BY group and summarized over all observations in the data table that is specified in the DATA= option.

```
/* Copy data to a CAS Table */
data mycas.air(replace=yes);
   set sashelp.air;
run;

data mycas.air(promote=yes);
set sashelp.air;
do class=1 to 10;
   output;
```

The following TSINFO procedure statements generate data tables that summarize data that have four BY groups:

```
proc tsinfo data=mycas.air outintervaldetails=mycas.interval2 nthreads=2;
  id date align=B;
```

```
  by class;
run;
```

The summarized information shows that BY groups 1 to 10 in the mycas.interval2 data table contain the interval fit information of the time ID values and shows the number of counts, alignment, and seasonality for each by group. All BY groups conform exactly to the MONTH interval. The information can be useful in diagnosing time series that contain many time intervals.

Chapter 5
The TSMODEL Procedure

Contents

Overview: TSMODEL Procedure **20**
- Comparison of the TSMODEL and TIMESERIES Procedures 20
- Comparison of the TSMODEL and TIMEDATA Procedures 20
- Using CAS Sessions and CAS Engine Librefs 23

Getting Started: TSMODEL Procedure **24**

Syntax: TSMODEL Procedure **26**
- Functional Summary 27
- PROC TSMODEL Statement 28
- BY Statement 32
- ID Statement 33
- INSCALARS Statement 39
- OUTARRAYS Statement 40
- OUTSCALARS Statement 40
- PRINT Statement 40
- VAR Statement 41
- REQUIRE Statement 41
- SUBMIT Statement 42
- ENDSUBMIT Statement 43
- SAS Programming Statements 43

Details: TSMODEL Procedure **43**
- Accumulation 44
- Missing Value Interpretation 45
- Summary Statistics 45
- SAS Programming Statements 45
- Predefined Symbols 46
- Auxiliary Tables 46
- Table Output 50
- Printed Output 53

Examples: TSMODEL Procedure **53**
- Example 5.1: Accumulating Transactional Data into Time Series Data 53
- Example 5.2: User-Defined Program Statements 55
- Example 5.3: Using Auxiliary Tables 56

References **59**

Overview: TSMODEL Procedure

The TSMODEL procedure is a SAS Viya procedure that executes user-defined programs on time series data. The TSMODEL procedure analyzes timestamped transactional data with respect to time and accumulates the data into a time series format.

The TSMODEL procedure forms time series from input timestamped transactional data and writes the accumulated time series variables to an output table. Time series are delineated based on the distinct values of the variables that are listed in the BY statement.

Timestamped transactional data are recorded at no fixed interval. Analysts often want to use time series analysis techniques that require fixed-time intervals. Therefore, the transactional data must be accumulated to form a fixed-interval time series, such as daily, weekly, or monthly.

The TSMODEL procedure forms time series vectors from timestamped data and then provides these vectors as array variables for subsequent processing by your program statements. Your program statements are processed independently for each BY group. The TSMODEL procedure is similar to the SAS DATA step for time series data. The SAS DATA step processes data by each row, whereas the TSMODEL procedure processes time series vectors for the BY groups.

Because PROC TSMODEL runs on SAS Cloud Analytic Services (CAS), it can process the time series data in parallel. Time series for the BY groups are divided across the nodes of the CAS session, and then threads are used on each node to process the node's BY groups concurrently.

All results of the time series analysis can be stored to CAS tables.

Comparison of the TSMODEL and TIMESERIES Procedures

The TSMODEL procedure has similarities to the TIMESERIES procedure in SAS/ETS. The TIMESERIES procedure enables you to perform a variety of standard time series analysis techniques with its various statements, whereas the TSMODEL procedure enables you to define your own analyses via user-defined program statements that you include in the procedure's statement block. The TSMODEL procedure provides no built-in time series analysis capabilities. You must provide a program to analyze the time series data. For more information about PROC TIMESERIES, see *SAS/ETS User's Guide*.

Comparison of the TSMODEL and TIMEDATA Procedures

The syntax of the TSMODEL procedure is similar to the syntax of the TIMEDATA procedure in SAS/ETS software. For more information about PROC TIMEDATA, see *SAS/ETS User's Guide*.

PROC TSMODEL requires that you specify CAS tables for all input data and all output data. Unlike PROC TIMEDATA, no actual processing of the time series data occurs in PROC TSMODEL; data in PROC TSMODEL are processed in the CAS session that is associated with the CAS librefs that you specify in the PROC TSMODEL statement.

Like PROC TIMEDATA, PROC TSMODEL supports auxiliary input tables, which you specify in the AUXDATA= option. However, PROC TSMODEL requires that either all or none of the BY variables are

present in an AUXDATA= table. No partial BY group matching for the AUXDATA= tables is supported at this time.

A simple example provides insight about the relative ease of moving from PROC TIMEDATA to PROC TSMODEL. Consider the following PROC TIMEDATA example, which rescales the Sale and Price variables for each of the Product BY groups:

```
proc timedata data=mylib.pricex
    outsum=mylib.pricexsum
    outarray=mylib.pricexoa
    outscalar=mylib.pricexos;
by Product;
id date interval=month start='01jan1998'd end='01dec2002'd;
var Sale / accumulate=total;
var Price / accumulate=avg;
outarray relsale relprice;
outscalar sbase pbase;
sbase=Sale[1];
pbase=Price[1];
do i=1 to _length_;
  if Sale[i] ne . then do;
    relsale[i] = Sale[i]/sbase;
  end;
  if Price[i] ne . then do;
     relprice[i]=Price[i]/pbase;
  end;
end;
run;
```

When you convert the preceding statements to use PROC TSMODEL with a CAS table, the code looks like this:

```
proc tsmodel data=mycas.pricex
     outsum=mycas.pricexsum
     outarray=mycas.pricexoa
     outscalar=mycas.pricexos;
by Product;
id date interval=month start='01jan1998'd end='01dec2002'd;
var Sale /accumulate=total;
var Price /accumulate=avg;
outarray relsale relprice;
outscalar sbase pbase;
submit;
sbase=Sale[1];
pbase=Price[1];
do i=1 to _length_;
  if Sale[i] ne . then do;
     relsale[i] = Sale[i]/sbase;
  end;
  if Price[i] ne . then do;
     relprice[i]=Price[i]/pbase;
  end;
end;
endsubmit;
quit;
```

Examine these two examples closely to see the differences. At first glance, the differences might not be very obvious. Although the differences in code syntax are subtle, the underlying differences in the procedure processing are profound.

One important difference between PROC TIMEDATA and PROC TSMODEL is that PROC TSMODEL requires you to use the SUBMIT and ENDSUBMIT statements to begin and end the set of programming statements that you want to execute on the time series BY groups. You can include SAS macro statements in the statements in your SUBMIT/ENDSUBMIT block. Table 5.1 shows the differences between the processing of programming statements in PROC TSMODEL compared to PROC TIMEDATA.

Table 5.1 Comparison of TSMODEL and TIMEDATA

Processing Step	TSMODEL	TIMEDATA
Statement parsing	Any parse-time errors are reported only after the ENDSUBMIT statement. Those errors come from the parsing and compilation of your code in your CAS session. SAS line numbers for incorrect statements are not reported in the SAS log.	Statements are parsed as the procedure works its way through its statement block. Any parse-time errors are interspersed with the program statements in the SAS log.
PROC FCMP subroutines and functions	Does not support as many built-in PROC FCMP functions and subroutines.	Supports built-in PROC FCMP functions and subroutines.
RUN_MACRO	Not supported because it cannot be performed by the CAS server when it executes your program statements in the context of your CAS session.	Can call the PROC FCMP subroutine RUN_MACRO to run a SAS macro, which can then call a SAS procedure.
ID statement with a SAS date or datetime variable	An ID variable is required because the value of the ID variable is needed to determine the time index of each row within the BY group. The accumulation frequency that you specify in the INTERVAL= option in the ID statement is required.	An ID statement is not required.
Custom intervals	Custom intervals are not supported at this time.	Custom intervals are supported.

Table 5.1 *continued*

Processing Step	TSMODEL	TIMEDATA
Accumulation modes	ACCUMULATE=NONE, FIRST, LAST, NOBS, and MEDIAN are not supported.	ACCUMULATE=NONE, FIRST, LAST, NOBS, and MEDIAN are supported.
Output data	Can be printed and plotted through other SAS procedures.	Supports PRINT and PLOT options to display data.

Using CAS Sessions and CAS Engine Librefs

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. This section describes how to create a CAS session and set up a CAS engine libref that you can use to connect to the CAS session. It assumes that you have a CAS server already available; contact your system administrator if you need help starting and terminating a server. This CAS server is identified by specifying the host on which it runs and the port on which it listens for communications. To simplify your interactions with this CAS server, the host information and port information for the server are stored as SAS option values that are retrieved automatically whenever this CAS server needs to be accessed. You can examine the host and port values for the server at your site by using the following statements:

```
proc options option=(CASHOST CASPORT);
run;
```

In addition to starting a CAS server, your system administrator might also have created a CAS session and a CAS engine libref for your use. You can define your own sessions and CAS engine librefs that connect to the CAS server as shown in the following statements:

```
cas mysess;
libname mycas cas sessref=mysess;
```

The CAS statement creates the CAS session named mysess, and the LIBNAME statement creates the mycas CAS engine libref that you use to connect to this session. It is not necessary to explicitly name the CASHOST and CASPORT of the CAS server in the CAS statement, because these values are retrieved from the corresponding SAS option values.

If you have created the mysess session, you can terminate it by using the TERMINATE option in the CAS statement as follows:

```
cas mysess terminate;
```

For more information about the CAS and LIBNAME statements, see the section “Introduction to Shared Concepts” on page 7 in Chapter 3, “Shared Concepts.”

Getting Started: TSMODEL Procedure

This section outlines the use of the TSMODEL procedure and describes some of the analysis techniques that you can perform on timestamped transactional data.

Suppose that a bank wants to analyze the transactions that are associated with each of its customers over time. Further, suppose that the CAS table mycas.transactions contains four variables that are related to these transactions: Customer, Date, Withdrawals, and Deposits. The following examples illustrate possible ways to analyze these transactions by using the TSMODEL procedure.

The following statements accumulate the timestamped transactional data to form a daily time series based on the accumulated daily totals of each type of transaction (Withdrawals and Deposits). These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
proc tsmodel data=mycas.transactions
             out=mycas.timeseries(replace=yes)
             outarray=mycas.arrays(replace=yes);
   by Customer;
   id Date interval=day accumulate=total;
   var withdrawals deposits;
   outarrays Balance;
   submit;
   Balance[1] = deposits[1] - withdrawals[1];
   do t = 2 to _LENGTH_;
      Balance[t] = Balance[t-1] + (deposits[t] - withdrawals[t]);
   end;
   endsubmit;
quit;
```

The OUT= option requests that the resulting time series data for each customer be stored in the table mycas.timeseries. The OUTARRAY= option requests that the resulting time series data along with a newly created variable, Balance, be stored in the table mycas.arrays. Both tables are created in the CAS session's current caslib.

The INTERVAL=DAY option requests that the transactions be accumulated on a daily basis within each Customer according to the values of the ID variable Date. The ACCUMULATE=TOTAL option requests that the sum of the transactions be calculated. After the transactional data are accumulated into a time series format, the example code computes a daily balance for each customer. Many of the procedures provided with SAS software can be used to perform further processing on the resulting time series data. The tables that are produced by PROC TSMODEL can also be used as input to subsequent PROC TSMODEL steps.

The TSMODEL procedure prints a summary of the time series processing that is performed, as shown in Figure 5.1.

Figure 5.1 Summary of Time Series Processing

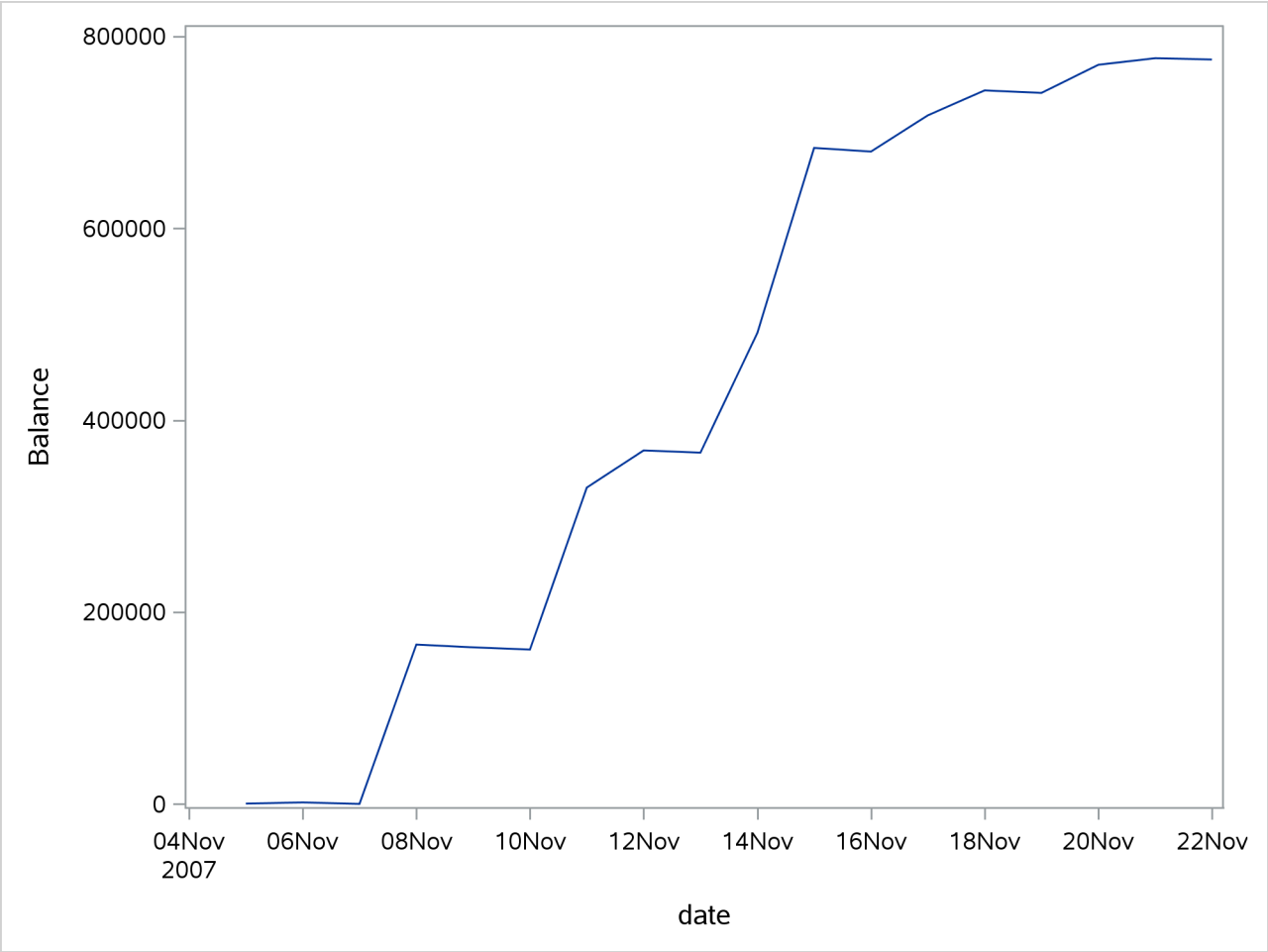
The TSMODEL Procedure

Summary of time series processing for TRANSACTIONS	
Number of analysis variables	2
Number of rows read	600
Number of groups read	3
Memory for group packages (KB)	4
Time to read groups (seconds)	0.0004930496
Time to load groups (seconds)	0.0049929619
Number of data threads	3
Thread BY group redundancy	1
Minimum time ID	05NOV2007
Maximum time ID	22NOV2007
Minimum time periods	18
Maximum time periods	18
Number of nodes run	1
Number of nodes with data	1
Number of nodes with groups	1
Number of threads budgeted	32
Minimum thread group count	0
Maximum thread group count	2
Minimum threads active	0
Maximum threads active	2
Average CPU utilization of nodes with data (%)	100
Minimum CPU utilization of nodes with data (%)	100
Maximum CPU utilization of nodes with data (%)	100
Number of groups processed by submitted code	3
Number of groups failing	0
Elapsed time to process groups (seconds)	0.0029840469
Number of output table rows produced	54
Number of array table rows produced	54

You might want to plot the generated Balance series for some particular customer. The following code produces a graph for the customer named 'Bill'; the graph is shown in Figure 5.2.

```
proc sgplot data=mycas.arrays(where=(customer='Bill'));
   series x=Date y=balance;
run;
```

Figure 5.2 Balance for Bill



Syntax: TSMODEL Procedure

The following statements are available in the TSMODEL procedure:

PROC TSMODEL *options* **;**
  BY *variables* **;**
  ID *variable* **INTERVAL=***interval* < *options* > **;**
  OUTARRAYS *array-name-list* **;**
  OUTSCALARS *scalar-name-list* **;**
  INSCALARS *scalar-name-list* **;**
  VAR *variable-list* / *options* **;**
  REQUIRE *package-list* **;**
  PRINT *print-options* **;**
  SUBMIT < **FILE=** *SAS-file-ref* | '*File-path*' > < *submit-options* > **;**
  Programming statements **;**
  ENDSUBMIT **;**

The PROC TSMODEL and ID statements are required.

The following sections present a summary of the statements and options that are used in PROC TSMODEL, a description of the TSMODEL statement, and then descriptions of the other statements.

Functional Summary

Table 5.2 summarizes the statements and options that control the TSMODEL procedure.

Table 5.2 Functional Summary

Description	Statement	Option
Statements		
Specifies BY-group processing	BY	
Specifies variables to analyze	VAR	
Specifies the time ID variable	ID	
Specifies the arrays to output	OUTARRAYS	
Specifies the scalars to output	OUTSCALARS	
Specifies the scalars that are input	INSCALARS	
Specifies the packages to include	REQUIRE	
Specifies the beginning of program statements	SUBMIT	
Specifies the end of program statements	ENDSUBMIT	
Print results from program execution	PRINT	
Table Options		
Specifies the auxiliary input tables	PROC TSMODEL	AUXDATA=
Specifies the input table	PROC TSMODEL	DATA=
Specifies the output table	PROC TSMODEL	OUT=
Specifies the array output table	PROC TSMODEL	OUTARRAY=
Specifies the scalar output table	PROC TSMODEL	OUTSCALAR=
Specifies the summary statistics output table	PROC TSMODEL	OUTSUM=
Specifies the table to contain the BY-group message output log	PROC TSMODEL	OUTLOG=
Specifies whether messages are output to the OUTLOG= table by severity	PROC TSMODEL	LOGCONTROL=
Specifies the input table for scalar variables	PROC TSMODEL	INSCALAR=
Specifies a collector object for an output table	PROC TSMODEL	OUTOBJ=
Specifies a repeater object for an input table	PROC TSMODEL	INOBJ=
Accumulation and Seasonality Options		
Specifies the accumulation frequency	ID	INTERVAL=
Specifies the length of the seasonal cycle	PROC TSMODEL	SEASONALITY=

Table 5.2 *continued*

Description	Statement	Option
Specifies the interval alignment	ID	ALIGN=
Specifies the starting time ID value	ID	START=
Specifies the ending time ID value	ID	END=
Specifies the accumulation statistic	ID, VAR	ACCUMULATE=
Specifies how to interpret missing values	ID, VAR	SETMISSING=
Specifies the method for trimming BY groups	ID	TRIMID=
Miscellaneous Options		
Specifies the forecast horizon or lead used to extend the CAS table	PROC TSMODEL	LEAD=
Specifies the format for the time ID variable	ID	FORMAT=
Specifies the file that contains the user-defined program	SUBMIT	FILE=
Specifies variable values available to the user-defined program	SUBMIT	DYNAMICS=

PROC TSMODEL Statement

PROC TSMODEL *options* **;**

The PROC TSMODEL statement invokes the TSMODEL procedure. You can specify the following *options*:

AUXDATA=*CAS-libref.data-table*

names a table that contains auxiliary input data for the procedure to use for supplying time series variables. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 23. For more information, see the section "Auxiliary Tables" on page 46.

DATA=*CAS-libref.data-table*

names the input data table for PROC TSMODEL to use. The default is the most recently created data table. *CAS-libref.data-table* is a two-level name, where

CAS-libref refers to a collection of information that is defined in the LIBNAME statement and includes the caslib, which includes a path to the data, and a session identifier, which defaults to the active session but which can be explicitly defined in the LIBNAME statement. For more information about *CAS-libref*, see the section "Using CAS Sessions and CAS Engine Librefs" on page 23.

data-table specifies the name of the input data table.

INOBJ=(*object-name***=***CAS-libref.data-table* **...)**

specifies pairs, each of which binds a repeater object specified by *object-name* with an input table specified by *CAS-libref.data-table*. You can specify one or more object-table pairs as needed to associate the repeater objects that you declare in your user-defined program with their input tables. You must specify a binding for any repeater object that you declare in your program; otherwise, a parse-time error is generated when you submit the program and no execution occurs. Consider the following SAS code:

```
inobj=(inest=mycas.outest
       inspec=mycas.outspec)
```

This code binds the repeater objects named INEST and INSPEC to the CAS tables mycas.outest and mycas.outspec, respectively. In addition to the columns that are required to satisfy the built-in table schema of a particular repeater object, each specified table must have all or none of the BY variables of the primary DATA= table. When the INOBJ= table has none of the BY variables, all the CAS table rows are input.

Repeater objects are defined in various packages that use PROC TSMODEL as a method to input data that are required for each application. For more information about creating repeater objects for a package, see *SAS Visual Forecasting: Time Series Packages*. For more information about package access, see the section "REQUIRE Statement" on page 41.

INSCALAR=*CAS-libref.data-table*

specifies a table to supply scalar dynamic variables to be included and made accessible to your program code as it executes.

CAS-libref.data-table is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 23.

When you specify BY variables for the DATA= table, the INSCALAR= table must contain those BY variables. If you do not specify BY variables, then the INSCALAR= table is read unqualified for the BY variables across the CAS workers in the CAS session. In this case, only a single value for each variable is needed, and only a single row is required in the table. If the table contains multiple rows in this case, an error is generated when the procedure is called. If you specify this option, then you must also specify one or more INSCALARS statements to specify the variables that you want to include for your program to access.

If BY variables are specified in a BY statement, then the table specified in the INSCALAR= option must contain all the specified BY variables or none of them. If BY variables are present in the INSCALAR= table, then the values for the variables specified in the INSCALARS statement are input subject to BY-group processing. If BY variables are not present in the INSCALAR= table, then only a single value of each variable specified in the INSCALARS statement is input for all BY groups. It is an error for the INSCALARS= table to contain more than one value (row) for the variables if the table is not subject to BY-group processing. If the INSCALARS= table is subject to BY-group processing and multiple values (rows) are present for any BY group, then the value can be input from any row, leading to inconsistent results. For consistent results, you should prepare the INSCALAR= tables such that only a single value is input for each BY group.

LEAD=*n*

specifies the number of periods ahead to extend time series arrays for the variables in both the VAR and OUTARRAYS statements that are output to the CAS table. You can specify this option to accommodate a forecast lead or horizon when you are preparing time series data for forecasting.

The value of *n* is relative to the ending value of the input time ID for each BY group as specified by the TRIMID= option in the ID statement; it is not relative to the last nonmissing observation of a particular series. By default, LEAD=0.

LOGCONTROL=(*severity***=IGNORE | KEEP <** *severity***=IGNORE | KEEP***... >***)**

specifies pairs that define error severity and associated message disposition for the OUTLOG= option. You can specify multiple LOGCONTROL= options.

You can specify zero or more pairs. In these pairs, *severity* can take one of the following values:

NONE specifies messages that have no severity classification.

NOTE specifies messages whose severity classification is NOTE.

WARNING specifies messages whose severity classification is WARNING.

ERROR specifies messages whose severity classification is ERROR.

You can specify the following values to indicate the associated message disposition:

IGNORE ignores messages of the specified severity.

KEEP retains messages of the specified severity.

This option is applied only when you specify an OUTLOG= option; otherwise, it has no effect. By default, LOGCONTROL=(ERROR=KEEP) when the OUTLOG= option is specified. This default value retains only messages whose severity classification is ERROR and discards all others. However, the default behavior no longer applies when you specify the LOGCONTROL= option. For example, if you specify LOGCONTROL=(NONE=KEEP), then only messages that have no severity classification are retained and all others (NOTE, WARNING, and ERROR) are discarded.

OUT=*CAS-libref.data-table*

names the output table to contain the time series variables that are specified in the subsequent VAR statements.

CAS-libref.data-table is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 23.

If BY variables are specified, they are also included in this output table. The ID variable's fixed-interval time ID sequence is included in the OUT= CAS table. The time series variables are accumulated based on the INTERVAL= option and the variable's ACCUMULATE= option. The OUT= CAS table is particularly useful when you want to further analyze, model, or forecast the resulting time series with other SAS procedures.

OUTARRAY=*CAS-libref.data-table*

names the output table to contain the time series vectors that are specified in the VAR and OUTARRAYS statements. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 23.

This table also contains the variables that are specified in the BY, ID, and VAR statements in addition to the arrays that are specified in the OUTARRAYS statements.

OUTLOG=*CAS-libref.data-table*

names the output table to contain textual messages that are collected from the execution of the BY-group processing.

Messages captured for the BY group are subject to prefiltering by their severity based on the setting of the LOGCONTROL= option. If PUTTOLOG=YES is specified, then messages from the PUT programming statement are also included in this table. This table has rows only for BY groups that generate text messages. Messages that are related to the PROC TSMODEL syntax are not included in this table. Normally, this table contains no rows.

OUTOBJ=(*object-name***=***CAS-libref.data-table* **...)**

specifies pairs, each of which binds a collector object specified by *object-name* with an output table specified by *CAS-libref.data-table*.

CAS-libref.data-table is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 23.

You can specify one or more object-table pairs as needed to associate the collector objects that you declare in your user-defined program with their output tables. You must specify a binding for any collector object that you declare in your program; otherwise, a parse-time error is generated when you submit the program and no execution occurs. Consider the following SAS code:

```
outobj=(oss=mycas.saleoss1
        pest=mycas.salepest1
        ofor=mycas.salefor1
        oind=mycas.saleind1
        ostat=mycas.salestat1)
```

This code binds the collector objects named OSS, PEST, OFOR, OIND, and OSTAT to the tables named SALEOSS1, SALEPEST1, SALEFOR1, SALEIND1, and SALESTAT1, respectively,. These tables are all created using CAS-related context from the mycas libref.

Collector objects are defined in various packages that can be run by PROC TSMODEL. For more information about using packages, see *SAS Visual Forecasting: Time Series Packages*. For more information about package access, see the section "REQUIRE Statement" on page 41.

OUTSCALAR=*CAS-libref.data-table*

names the output table to contain the scalar names that are specified in the OUTSCALARS statements. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier,

and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 23.

This table also contains the variables that are specified in the BY statement and the scalars that are specified in the OUTSCALARS statements.

OUTSUM=*CAS-libref.data-table*

names the output table to contain the descriptive statistics. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the output data table. For more information about this two-level name, see the DATA= option and the section "Using CAS Sessions and CAS Engine Librefs" on page 23.

The descriptive statistics are based on the accumulated time series when the ACCUMULATE= option, the SETMISSING= option, or both are specified in the ID or VAR statements. This table is particularly useful when you want to analyze large numbers of series and you need a summary of the results.

PUTTOLOG=YES | NO

specifies whether to capture messages from the PUT programming statement to the BY group's row in the table that is specified in the OUTLOG= option. You can specify the following values:

NO does not capture messages.

YES captures messages.

This option is applied only when you specify the OUTLOG= option; otherwise, it has no effect. This option should be used only to aid in debugging your user-defined programs on small amounts of data. This option can produce large amounts of output that increases processing time and memory usage in your CAS session processes and should be used with caution. By default, PUTTOLOG=NO.

SEASONALITY=*number*

specifies the length of the seasonal cycle. For example, SEASONALITY=3 means that every group of three time periods forms a seasonal cycle. By default, the length of the seasonal cycle is 1 (no seasonality) or the length that is implied by the INTERVAL= option in the ID statement. For example, INTERVAL=MONTH implies that the length of the seasonal cycle is 12.

BY Statement

BY *variables* **;**

You can include a BY statement with PROC TSMODEL to obtain separate processing for groups of observations that are defined by the BY *variables*. The rows in each distinct BY group are used to accumulate the time series vectors from the timestamped row data according to a desired frequency (which is specified in the INTERVAL= option) and the accumulation mode (which is specified in the ACCUMULATE= option).

For more information about table output and BY group processing, see the section "Table Output" on page 50.

ID Statement

ID *variable* **INTERVAL=***interval* < *options* > **;**

The ID statement names a numeric *variable* that identifies the temporal order (time sequence) of observations in the input and output tables. The values of variables are assumed to be SAS date, SAS datetime, or observation values. In addition, the ID statement specifies the frequency to be associated with the time series. The ID statement *options* also specify a global treatment for how the time series variables are accumulated from the BY group's rows and how the time ID values are aligned to form the time series. The specified information affects all variables that are specified in subsequent VAR statements. The ID statement and the INTERVAL= option are required to specify the desired accumulation frequency.

You must specify the following option:

INTERVAL=*interval*

specifies the frequency of the accumulated time series. For example, if the input table consists of quarterly observations, then specify INTERVAL=QTR.

Interval names are constructed from a basic interval type with an optional multiplier and shift. The general form of an interval name is as follows:

type< *multiplier* >.< *shift* >

The interval *type* for SAS date values are summarized as follows:

YEAR

specifies yearly intervals. Abbreviations are YEAR, YEARS, YEARLY, YR, ANNUAL, ANNUALLY, and ANNUALS. The starting subperiod *shift* is in months (MONTH).

YEARV

specifies ISO 8601 yearly intervals. The ISO 8601 year starts on the Monday immediately preceding January 4 (or on January 4 if it is a Monday). Note that it is possible for the ISO 8601 year to start in December of the preceding year. Also, some ISO 8601 years contain a leap week. For more information about ISO weeks, see Technical Committee ISO/TC 154 (Processes, Data Elements, and Documents in Commerce, Industry, and Administration) (2004). The starting subperiod *shift* is in ISO 8601 weeks (WEEKV).

R445YR

is the same as YEARV except that the starting subperiod *shift* is in retail 4-4-5 months (R445MON).

R454YR

is the same as YEARV except that the starting subperiod *shift* is in retail 4-5-4 months (R454MON). For more information about the retail 4-5-4 calendar, see National Retail Federation (2007).

R544YR

is the same as YEARV except that the starting subperiod *shift* is in retail 5-4-4 months (R544MON).

SEMIYEAR

specifies semiannual intervals (every six months). Abbreviations are SEMIYEAR, SEMIYEARS, SEMIYEARLY, SEMIYR, SEMIANNUAL, and SEMIANN.

The starting subperiod *shift* is in months (MONTH). For example, SEMIYEAR.3 intervals are March–August and September–February.

QTR

specifies quarterly intervals (every three months). Abbreviations are QTR, QUARTER, QUARTERS, QUARTERLY, QTRLY, and QTRS. The starting subperiod *shift* is in months (MONTH).

R445QTR

specifies retail 4-4-5 quarterly intervals (every 13 ISO 8601 weeks). Some fourth quarters contain a leap week. The starting subperiod *shift* is in retail 4-4-5 months (R445MON).

R454QTR

specifies retail 4-5-4 quarterly intervals (every 13 ISO 8601 weeks). Some fourth quarters contain a leap week. For more information about the retail 4-5-4 calendar, see National Retail Federation (2007). The starting subperiod *shift* is in retail 4-5-4 months (R454MON).

R544QTR

specifies retail 5-4-4 quarterly intervals (every 13 ISO 8601 weeks). Some fourth quarters contain a leap week. The starting subperiod *shift* is in retail 5-4-4 months (R544MON).

MONTH

specifies monthly intervals. Abbreviations are MONTH, MONTHS, MONTHLY, and MON. The starting subperiod *shift* is in months (MONTH). For example, MONTH2.2 intervals are February–March, April–May, June–July, August–September, October–November, and December–January of the following year.

R445MON

specifies retail 4-4-5 monthly intervals. The 3rd, 6th, 9th, and 12th months are five ISO 8601 weeks long with the exception that some 12th months contain leap weeks. All other months are four ISO 8601 weeks long. R445MON intervals begin with the 1st, 5th, 9th, 14th, 18th, 22nd, 27th, 31st, 35th, 40th, 44th, and 48th weeks of the ISO year. The starting subperiod *shift* is in retail 4-4-5 months (R445MON).

R454MON

specifies retail 4-5-4 monthly intervals. The 2nd, 5th, 8th, and 11th months are five ISO 8601 weeks long. All other months are four ISO 8601 weeks long with the exception that some 12th months contain leap weeks. R454MON intervals begin with the 1st, 5th, 10th, 14th, 18th, 23rd, 27th, 31st, 36th, 40th, 44th, and 49th weeks of the ISO year. For more information about the retail 4-5-4 calendar, see National Retail Federation (2007). The starting subperiod *shift* is in retail 4-5-4 months (R454MON).

R544MON

specifies retail 5-4-4 monthly intervals. The 1st, 4th, 7th, and 10th months are five ISO 8601 weeks long. All other months are four ISO 8601 weeks long with the exception that some 12th months contain leap weeks. R544MON intervals begin with the 1st, 6th, 10th, 14th, 19th, 23rd, 27th, 32nd, 36th, 40th, 45th, and 49th weeks of the ISO year. The starting subperiod *shift* is in retail 5-4-4 months (R544MON).

SEMIMONTH

specifies semimonthly intervals. SEMIMONTH breaks each month into two periods, starting on the 1st and 16th days. Abbreviations are SEMIMONTH, SEMIMONTHS, SEMIMONTHLY, and SEMIMON. The starting subperiod *shift* is in SEMIMONTH periods. For example, SEMIMONTH2.2 specifies intervals from the 16th of one month through the 15th of the next month.

TENDAY

specifies 10-day intervals. TENDAY breaks the month into three periods, the 1st through the 10th day of the month, the 11th through the 20th day of the month, and the remainder of the month. (TENDAY is a special interval typically used for reporting automobile sales data.) The starting subperiod *shift* is in TENDAY periods. For example, TENDAY4.2 defines 40-day periods that start at the second TENDAY period.

WEEK

specifies weekly intervals of seven days. Abbreviations are WEEK, WEEKS, and WEEKLY. The starting subperiod *shift* is in days (DAY), with the days of the week numbered as 1=Sunday, 2=Monday, 3=Tuesday, 4=Wednesday, 5=Thursday, 6=Friday, and 7=Saturday. For example, WEEK.7 means weekly with Saturday as the first day of the week.

WEEKV

specifies ISO 8601 weekly intervals of seven days. Each week starts on Monday. The starting subperiod *shift* is in days (DAY). Note that WEEKV differs from WEEK in that WEEKV.1 starts on Monday, WEEKV.2 starts on Tuesday, and so on.

WEEKDAY

WEEKDAY*d*W

WEEKDAY*dd*W

WEEKDAY*ddd*W

specifies daily intervals with weekend days included in the preceding weekday. Note that for a five-day work week that starts on Monday, the appropriate interval is WEEKDAY5.2. Abbreviations are WEEKDAY and WEEKDAYS. The starting subperiod *shift* is in weekdays (WEEKDAY).

The WEEKDAY interval is the same as DAY except that weekend days are absorbed into the preceding weekday. Thus, there are five WEEKDAY intervals in a calendar week: Monday, Tuesday, Wednesday, Thursday, and the three-day period Friday-Saturday-Sunday.

The default weekend days are Saturday and Sunday, but any one to six weekend days can be listed after the WEEKDAY string and followed by a W. Weekend days are specified as ‘1’ for Sunday, ‘2’ for Monday, and so forth. For example, WEEKDAY67W specifies a Friday-Saturday weekend. WEEKDAY1W specifies a six-day work week with a Sunday weekend. WEEKDAY17W is the same as WEEKDAY.

DAY

specifies daily intervals. Abbreviations are DAY, DAYS, and DAILY. The starting subperiod *shift* is in days (DAY).

The interval *type* for SAS datetime values are summarized as follows:

DT*SAS-date-interval*
specifies a SAS datetime interval that corresponds to the SAS date interval but operates on SAS datetime values. The SAS datetime interval is created by adding the prefix "DT" to the SAS date interval name. For example, DTMONTH.

HOUR
specifies hourly intervals. Aliases are HOUR, DTHOUR, HOURS, DTHOURS, HOURLY, DTHOURLY, HR, and DTHR. The starting subperiod *shift* is in hours (HOUR).

MINUTE
specifies minute intervals. Aliases are MINUTE, DTMINUTE, MINUTES, DTMINUTES, MIN, and DTMIN. The starting subperiod *shift* is in minutes (MINUTE).

SECOND
specifies second intervals. Aliases are SECOND, DTSECOND, SECONDS, DTSECONDS, SEC and DTSEC. The starting subperiod *shift* is in seconds (SECOND).

The interval *type* for SAS observation values is summarized as follows:

OBS

n
specifies that observation numbers identify the time periods. You can specify INTERVAL=*n* as an alias for INTERVAL=OBS*n*.

If the SEASONALITY= option is not specified in the PROC TSMODEL statement, then the length of the seasonal cycle is implied from the INTERVAL= option. For example, INTERVAL=QTR implies a seasonal cycle of length 4. If the ACCUMULATE= option is also specified, the INTERVAL= option determines the time periods for the accumulation of observations. The INTERVAL= option is required and must be specified in the ID statement.

You can also specify the following *options*:

ACCUMULATE=*option*
specifies how to accumulate the table observations within each time period into a single value. The frequency (width of each time interval) is specified in the INTERVAL= option. The ID *variable* contains the time ID values. The value of the time ID *variable* identifies the time period of the observation. The accumulated values for each time period form the time series, which is used in subsequent analysis.

The ACCUMULATE= option is necessary when multiple input observations identify the same time period. An example of this is timestamped transactional data.

The following *options* determine how to accumulate the observations within each time period based on the ID variable and the frequency specified by the INTERVAL= option. Each value indicates how the accumulated value is calculated:

TOTAL | SUM	calculates as the total sum of the nonmissing values.
AVERAGE | AVG	calculates as the average of the nonmissing values.
MINIMUM | MIN	calculates as the minimum of the nonmissing values.
MAXIMUM | MAX	calculates as the maximum of the nonmissing values.
N	calculates as the number of nonmissing observations.

NMISS calculates as the number of missing observations.

STDDEV | STD calculates as the standard deviation of the nonmissing values.

CSS calculates as the corrected sum of squares of the nonmissing values.

USS calculates as the uncorrected sum of squares of the nonmissing values.

By default, ACCUMULATE=TOTAL.

The SETMISSING= option is useful for specifying how to treat missing values in the accumulated time series variable. If missing values should be interpreted as 0, then specify SETMISSING=0. For more information about accumulation, see the section "Accumulation" on page 44.

ALIGN=*option*

controls the alignment of SAS date or datetime values that are used to identify the time period of output observations. Although any date or datetime value within the time period can identify the time period, the ALIGN= option requests that the representative date or datetime for the time period be calculated as the beginning date or datetime of the time period, the ending date or datetime of the time period, or the middle date or datetime of the time period. In addition to aligning the time ID values consistently for observations that are supplied by the user, the ALIGN= option specifies the method for calculating the time ID values for observations in the forecast and backcast time periods, which often are not supplied by the user. You can specify the following options:

BEGINNING | BEG | B represents each time period using the beginning SAS date or datetime value of the time period.

ENDING | END | E represents each time period using the ending SAS date or datetime value of the time period.

MIDDLE | MID | M represents each time period using the middle SAS date or datetime value of the time period. The middle is calculated as the average of the beginning and ending values.

By default, ALIGN=BEGINNING.

END=*value*

specifies a SAS date or datetime value that represents the end of the data. If the last time ID variable value is less than *value*, then the series is extended with missing values. If the last time ID variable value is greater than *value*, then the series is truncated. For example, END="&sysdate"D uses the automatic macro variable SYSDATE to extend or truncate the series to the current date. You can specify the START= and END= options to ensure that the data that are associated within each BY group contain the same number of observations.

NOTE: When you do not specify both the START= and END= options in the ID statement, the TSMODEL procedure must make an additional pass through the data set that you specified in the DATA= option in order to determine the data set's time index span so that it can compute the total number of accumulation time intervals. In this scenario, the TSMODEL procedure reads all rows from the data set twice. This effect is captured in the numeric value that is shown in the row labeled 'Number of rows read' in the time series processing summary table that is generated at the end of the execution of the TSMODEL procedure, as is illustrated in Figure 5.1 and Output 5.1.1.

FORMAT=*format*

specifies the SAS format for the time ID values. If this option is not specified, the default format is inferred from the INTERVAL= option.

SETMISSING=*option*

specifies how to interpret missing values (either actual or accumulated) in the accumulated time series. You can use the following *options* to determine how to interpret missing values:

n	interprets a missing value as having the value *n*. You can specify any number for *n*, but not a missing value. If a missing value indicates a 0 value, specify SETMISSING=0. You typically use SETMISSING=0 for transactional data because no recorded data usually implies no activity.
MISSING	interprets a missing value as a missing value. Use this option if a missing value indicates an unknown value.
AVERAGE | AVG	interprets a missing value as the average value of all accumulated nonmissing values in the span of the series.
MINIMUM | MIN	interprets a missing value as the minimum value of all accumulated nonmissing values in the span of the series.
MEDIAN | MED	interprets a missing value as the median value of all accumulated nonmissing values in the span of the series.
MAXIMUM | MAX	interprets a missing value as the maximum value of all accumulated nonmissing values in the span of the series.
FIRST	interprets a missing value as the first nonmissing value of all accumulated nonmissing values in the span of the series.
LAST	interprets a missing value as the last nonmissing value of all accumulated nonmissing values in the span of the series.
PREVIOUS | PREV	interprets a missing value as the previous period's accumulated nonmissing value. Missing values at the beginning of the accumulated series remain missing.
NEXT	interprets a missing value as the next period's accumulated nonmissing value. Missing values at the end of the accumulated series remain missing.

By default, SETMISSING=MISSING.

START=*value*

specifies a SAS date or datetime value that represents the beginning of the data. If the first time ID variable value is greater than *value*, then missing values are added at the beginning of the series. If the first time ID variable value is less than *value*, then the series is truncated. You can specify the START= and END= options to ensure that data associated with each BY group contain the same number of observations.

NOTE: When you do not specify both the START= and END= options in the ID statement, the TSMODEL procedure must make an additional pass through the data set that you specified in the DATA= option in order to determine the data set's time index span so that it can compute the total number of accumulation time intervals. In this scenario, the TSMODEL procedure reads all rows from the data set twice. This effect is captured in the numeric value that is shown in the row labeled 'Number

of rows read' in the time series processing summary table that is generated at the end of the execution of the TSMODEL procedure, as is illustrated in Figure 5.1 and Output 5.1.1.

TRIMID=*method*

specifies the method for trimming the data in the BY groups when time series vectors are input to the user-defined program. The output time ID variable span that is calculated by the *method* is dependent on the input time ID variable span, irrespective of missing values of the time series variables. Depending on the *method* and the input time ID variable data, leading or trailing missing values can be added to the time series variables.

After the output time ID variable span is calculated by the *method*, the ending value of the output time ID variable will be recalculated according to the value of the LEAD= option (if one is specified).

You can specify one of the following *methods*:

NONE uses the same starting and ending values of the output time ID variable for all BY groups. The span of the output time ID variable includes all values that are input as a time ID value for all BY groups. The time series variables are extended with leading or trailing missing values as required.

LEFT uses the identifying date for the first time period that is input for the BY group as the starting value of the output time ID variable for each BY group. The time series values in each BY group are not extended with leading missing values. The ending value of the time ID variable is the same for all BY groups. The time series variables are extended with trailing missing values as required.

RIGHT uses the identifying date for the last time period that is input for the BY group as the ending value of the output time ID variable for each BY group. The time series values in each BY group are not extended with trailing missing values. The starting value of the time ID variable is the same for all BY groups. The time series variables are extended with leading missing values as required.

BOTH uses the span of the input time ID variable for the BY group as the span of the output time ID variable for each BY group. The time series values in each BY group are not extended with leading or trailing missing values.

By default, TRIMID=NONE.

INSCALARS Statement

INSCALARS *scalar-name-list* **;**

INSCALAR *scalar-name-list* **;**

The INSCALARS statement specifies which scalar variables to automatically include for your program to use. These variables can be numeric or character. Each variable that you name in an INSCALARS statement must be defined in the table specified in the INSCALAR= option in the PROC TSMODEL statement. You must specify at least one INSCALARS statement if you specify the INSCALAR= option in the PROC TSMODEL statement. You can specify multiple INSCALARS statements. For more information, see the INSCALAR= option.

OUTARRAYS Statement

OUTARRAYS *array-name-list* **;**

OUTARRAY *array-name-list* **;**

The OUTARRAYS statement specifies a list of array names; each name specifies a numeric output array variable to be stored in the table that is specified in the OUTARRAY= option in the PROC TSMODEL statement. You can specify multiple OUTARRAYS statements.

Your program statements can create and use any number of arrays. The array variables that are specified in the OUTARRAYS statement are created automatically for you, and their lengths are predetermined by the length of the time ID vector for the BY group. The arrays are initialized to missing values at the start of each BY group. Although you can define and use many arrays in your program, only arrays that are listed in the OUTARRAYS statement are included in the table that is specified in the OUTARRAY= option.

OUTSCALARS Statement

OUTSCALARS *scalar-name-list* **;**

OUTSCALAR *scalar-name-list* **;**

The OUTSCALARS statement specifies a list of scalar names; each name specifies a numeric or character output scalar variable to be stored in the table that is specified in the OUTSCALAR= option in the PROC TSMODEL statement. You can specify multiple OUTSCALARS statements.

Your program statements can create and use any number of scalars. Only scalars that are listed in the OUTSCALARS statement are predefined and included in the table that is specified in the OUTSCALAR= option. The scalars are initialized to missing values at the start of each BY group. Each scalar name can specify numeric or character scalar variables which are defined as follows:

var-list defines the variables in the *var-list* as numeric scalars.

var-list $*n* defines the variables in the *var-list* as character scalars with length *n*.

The following statement defines scalar variables rc, holdout, and nchanges as numeric scalars, and defines selected as character with length 32.

```
outscalar rc selected $32 holdout nchanges;
```

PRINT Statement

PRINT *print-options* **;**

The PRINT statement allows you to display results that are saved to CAS tables that you specify in the PROC TSMODEL statement options. You can include multiple PRINT statements in your PROC TSMODEL statement block. Each PRINT statement is processed independently in the order that you specify them.

The PRINT OUTLOG statement has the following form:

PRINT OUTLOG < **WHERE** *where-clause* > **;**

PRINT OUTLOG displays information from the OUTLOG= table to the SAS LOG. If you specify a BY statement, then the messages that are stored to the OUTLOG table are printed in BY group order followed with a BY line separator. You can specify an optional WHERE clause to select the rows from the OUTLOG table that you want to print. The WHERE clause must reference columns of the OUTLOG= table.

Following are four examples of PRINT OUTLOG statements that you might specify:

```
print outlog;
print outlog where regionName eq 'Region1';
print outlog where _errno_ ne 0;
print outlog where _log_ contains "sale";
```

The first PRINT statement prints all OUTLOG= table rows to the SAS LOG, the second selects rows where the regionName column has the value 'Region1', the third selects rows where the _errno_ column has a non-zero value, and the last selects rows where the _log_ column text contains the word 'sale'.

VAR Statement

VAR *variable-list* < / *options* > **;**

The VAR statement lists the numeric variables in the DATA= table or AUXDATA= tables whose values are to be accumulated to form the time series.

An input table variable can be specified in only one VAR statement. You can specify any number of VAR statements. You can also specify the following *options* in the VAR statements:

ACCUMULATE=*option*
: specifies how to accumulate the table observations within each time period for the variables in the *variable-list*. If you do not specify this option, accumulation is determined by the ACCUMULATE= option in the ID statement.

SETMISS=*option* | *number*

SETMISSING=*option* | *number*
: specifies how to interpret missing values (either actual or accumulated) in the accumulated time series for variables in the *variable-list*. If you do not specify this option, missing values are set based on the SETMISSING= option in the ID statement.

REQUIRE Statement

REQUIRE *package-1* < *package-2 ... package-n* > **;**

REQUIRE *package (class-1 ...class-n)* **;**

The REQUIRE statement specifies which time series and time frequency analysis packages to make available for your user-defined program. These packages include functions, subroutines, and objects that you can utilize from your program to perform sophisticated time series processing. These packages provide functionality that ranges from a simple function to count missing observations in an array to very sophisticated objects that perform automatic time series modeling and forecasting.

The first form of the REQUIRE statement enables you to specify a list of package names that are available for use. If you use this form, you must specify at least one package name. You can specify multiple packages in the same REQUIRE statement.

The second form of the REQUIRE statement enables you to selectively identify which objects in *package* you intend to use. Packages might contain a number of different objects, and it can be more efficient to register only the ones that you plan to use.

You can specify multiple REQUIRE statements. All packages that are specified in REQUIRE statements are loaded prior to parsing your program statements so that any references are defined at the time your code is parsed. If you specify an invalid package name, then an error is returned prior to parsing your program statements. For more information, see *SAS Visual Forecasting: Time Series Packages*.

SUBMIT Statement

SUBMIT < **FILE=** *SAS-file-ref* | '*File-path*' > < *submit-options* > **;**

The SUBMIT statement indicates the start of your user-defined program statements or specifies the location of a file that contains your user-defined program statements.

If the SUBMIT statement does not specify a file, then all lines of code that are specified between the SUBMIT statement and the ENDSUBMIT statement are executed on the time series BY groups that arise from your timestamped input data. Compilation errors are reported back to the SAS log, and no further processing occurs.

Notice the placement of the SUBMIT and ENDSUBMIT in the example in the "Comparison of the TSMODEL and TIMEDATA Procedures" on page 20. The use of SUBMIT and ENDSUBMIT statements to delineate of the program block is a major difference between PROC TSMODEL and PROC TIMEDATA.

You can specify the following *submit-options*:

FILE=*SAS-file-ref* | '*File-path*'

specifies a file that contains your user-defined program. If you specify this option, you must not specify the ENDSUBMIT statement.

You can specify the file either by including a *SAS-file-ref* that is defined in a prior SAS FILENAME statement or by including a quoted string that identifies the host-specific file path.

The following rules apply when you include the FILE= option:

- The text file you specify must contain only the program statements that you want to submit and none of the PROC TSMODEL statements. You must not include an ENDSUBMIT statement in the file.
- The text file you specify must not contain any SAS macro statements or macro variable references. The file content is submitted as specified.
- You are free to use any mechanism of your choosing to generate the program statements in the text file. You can use a SAS DATA step with PUT statements, PROC LUA, PROC STREAM, and so on to generate the file from within a SAS program. You can use your favorite editor to generate the text file and then specify it in the SUBMIT statement if the content is static. For more information, see *Getting Started with SAS Viya for Lua*.

DYNAMICS=(*var-1***=[***number* | *"string"***] <** *var-2***=[***number* | *"string"***]... >)**

defines variables (constants) outside of the scope of the submitted code. These variables function as global arguments to the submitted code. In conjunction with the INSCALARS statement, these values can be used and evaluated within the submitted code. You can specify both numeric and character variable types. The type of the variable is determined by the right-hand side of the variable's declaration. For example, consider the following DYNAMICS= option:

```
DYNAMICS=(SOF='MAPE' HOLDOUT=10)
```

The variable SOF is defined as a character variable, and its static value is 'MAPE'. The variable HOLDOUT is defined as a numeric variable, and its static value is 10.

ENDSUBMIT Statement

ENDSUBMIT ;

The ENDSUBMIT statement is required to terminate the user-defined program statements in the PROC TSMODEL statement block whenever you specify the SUBMIT statement without the FILE= option. Do not specify the ENDSUBMIT statement if you specify the FILE= option in the SUBMIT statement. If you do not specify the FILE= option and you fail to include an ENDSUBMIT statement to accompany your SUBMIT statement, then the PROC TSMODEL step ends with an error.

SAS Programming Statements

Programming statements ;

You can use all the programming statements that are allowed in PROC FCMP except the following:

- CALL RUN_MACRO subroutine
- CALL READ_ARRAY subroutine
- CALL WRITE_ARRAY subroutine
- Custom intervals in the INTCK and INTNX functions

Details: TSMODEL Procedure

The TSMODEL procedure forms time series data from transactional data, which are analyzed according to the following steps if the relevant option listed on the right is specified:

1. accumulation	ACCUMULATE= option in the ID or VAR statement
2. missing value interpretation	SETMISSING= option in the ID or VAR statement
3. program execution	user-defined program statements
4. descriptive statistics	OUTSUM= option

Accumulation

If you specify the ACCUMULATE= option in the ID or VAR statement, observations in the table that is specified in the DATA= option in the PROC TSMODEL statement are accumulated within each time period. The frequency (width of each time interval) is specified by the INTERVAL= option in the ID statement. The ID variable contains the time ID values. Each time ID value is a SAS date or datetime that identifies the time period that contains the date or datetime. Accumulation is useful when the input table contains transactional data, whose observations are not spaced with respect to any particular time interval. The accumulated values form the time series, which is used in subsequent analyses.

For example, suppose an input table contains the following observations:

```
19MAR1999    10
19MAR1999    30
11MAY1999    50
12MAY1999    20
23MAY1999    20
```

If INTERVAL=MONTH is specified in the ID statement, then all the preceding observations fall within a three-month period of time between March 1999 and May 1999. The observations are accumulated within each time period as follows:

- If the ACCUMULATE=TOTAL option is specified, the resulting time series is

```
01MAR1999    40
01APR1999    .
01MAY1999    90
```

- If the ACCUMULATE=AVERAGE option is specified, the resulting time series is

```
01MAR1999    20
01APR1999    .
01MAY1999    30
```

- If the ACCUMULATE=MINIMUM option is specified, the resulting time series is

```
01MAR1999    10
01APR1999    .
01MAY1999    20
```

- If the ACCUMULATE=MAXIMUM option is specified, the resulting time series is

```
01MAR1999    30
01APR1999    .
01MAY1999    50
```

- If the ACCUMULATE=STDDEV option is specified, the resulting time series is

```
01MAR1999    14.14
01APR1999    .
01MAY1999    17.32
```

As you can see from the preceding examples, the accumulated time series can have missing values even though the input table observations contain no missing values.

Missing Value Interpretation

Once the data has been accumulated to form a time series based on the INTERVAL= and the ACCUMULATE= options, missing value interpretation is performed. Sometimes missing values should be interpreted as unknown values. In other situations, missing values are known, such as when missing values are created from accumulation and no observation should be interpreted as no value—that is, 0. In the former case, you can specify the SETMISSING= option to interpret how to treat missing values. In the latter case, you can specify SETMISSING=0 in order to treat missing observations as no (zero) values. In other cases, missing values should be interpreted as global values, such as minimum or maximum values of the accumulated series. The accumulated and interpreted time series is used in subsequent analyses.

Summary Statistics

You can compute summary statistics from the working series by specifying the OUTSUM= option in the PROC TSMODEL statement.

SAS Programming Statements

The user-defined program for PROC TSMODEL can contain most of the SAS programming statements and functions available in the DATA step or in the FCMP procedure. However, there are a few differences as noted in "SAS Programming Statements" on page 43. For more information, see the "FCMP Procedure" chapter in the *SAS Visual Data Management and Utility Procedures Guide*.

All variables that are specified in the ID and VAR statements are assigned as predefined arrays for subsequent processing. In addition, all array names that are specified in the OUTARRAYS statements and all the scalars names that are specified in the OUTSCALARS statements are assigned as predefined symbols for subsequent processing.

Predefined Symbols

In addition to the predefined arrays that are specified in the OUTARRAYS statements and the predefined scalars that are specified in the OUTSCALARS statements, the TSMODEL procedure creates predefined symbols that are automatically available for use in the programming statements. The name and description of the predefined symbols are shown in the following subsections; these names must not be used as variable names in any input data set.

Predefined Scalar Values

FORMAT	time format, which is either implied by the INTERVAL= option or specified in the FORMAT= option in the ID statement
HORIZON	the time ID value that is one period beyond _TEND_
INTERVAL	time interval, which is specified in the INTERVAL= option in the ID statement
LEAD	forecast horizon or lead, which is specified in the LEAD= option in the PROC TSMODEL statement
LENGTH	length of the time series that is associated with the current BY group
SERIES	series index or BY-group counter. When PROC TSMODEL runs in a parallel processing environment, the index value is relative to the machine that processes the series or BY group. Therefore, values of _SERIES_ might not be unique and could also vary for multiple executions of the procedure.
SEASONALITY	length of the seasonal cycle, which is specified in the SEASONALITY= option in the PROC TSMODEL statement or implied by the INTERVAL= option in the ID statement
TEND	the last time ID value over all the BY groups. _TEND_ is independent of the value of the TRIMID= option in the ID statement, but the value of _TEND_ is the same as the ending value of the output time ID variable span if you specify TRIMID=NONE.
TSTART	the first time ID value over all the BY groups. _TSTART_ is independent of the value of the TRIMID= option in the ID statement, but the value of _TSTART_ is the same as the starting value of the output time ID variable span if you specify TRIMID=NONE.

Predefined Array Values

SEASON	season index values
CYCLE	life-cycle index values

Auxiliary Tables

The TSMODEL procedure can use auxiliary tables to contribute input variables to the run of the procedure step. This functionality creates a virtual data source that allows some of the input variables to physically

reside in different tables. Some input variables can reside in the primary table, which is specified in the DATA= option, and other input variables can reside in the tables that are specified in one or more AUXDATA= options. This functionality enables sharing of common time series data across multiple projects.

You can specify more than one auxiliary data source to be used to input time series vectors across a particular BY-group hierarchy. To simplify data management, you can isolate variables that have naturally different levels of BY-group qualification into separate tables and use separate AUXDATA= options to supply them.

AUXDATA Functionality

There are two classes of time series table sources:

- a primary table from the DATA= option
- auxiliary data sources from AUXDATA= options

The AUXDATA= option specifies an auxiliary table that provides time series variables that are required for processing but are not included in the table that is specified in the DATA= option.

You can specify multiple AUXDATA= options in the PROC TSMODEL statement. Each AUXDATA= option establishes an auxiliary table source to supply variables that are declared in subsequent statements in the procedure step. If no auxiliary data sources are required, then the AUXDATA= option can be omitted.

Variables referenced in the TSMODEL procedure fall into three classes:

- variables that must be physically present in the primary table, which is specified in the DATA= option
- variables that must be physically present in each auxiliary table that is specified in an AUXDATA= option
- variables that can reside in either the primary or an auxiliary table

The ID variable for PROC TSMODEL must be present in the primary table and all the auxiliary tables that you specify. Variables that you specify in the BY statement must be present in the primary table. The auxiliary tables must contain all of those BY variables or none of them. Partial matching of a leftmost subset of the BY variables is not supported for the auxiliary tables.

The time series variables that you specify in VAR statements can be input from either the primary table or an auxiliary table. Variable resolution proceeds in reverse order from the last AUXDATA= option in the PROC TSMODEL statement to the first. If the variable in question is not found in any of those, the variable must be present in the primary table for the procedure step to be successful.

AUXDATA Alignment across BY Groups

All variables in the BY statement must be physically present in the primary table. However, it is not necessary to have the BY variables present in any of the auxiliary tables. All or none of the BY variables can be present in any auxiliary table.

For example, suppose you have a hierarchy of (REGION, PRODUCT) in the primary table Sales, which holds the time series variables for monthly sales metrics. Suppose you have an auxiliary table called Promotions that has no BY variables and contains analysis variables for promotions, and another table called Returns

that contains time series analysis variables for (REGION, PRODUCT) level groupings. In this scenario, each (REGION, PRODUCT) group in the Sales table always includes the time series variables from the Promotions table, and the analysis variables for the matching (REGION, PRODUCT) BY groups from the Returns table. So if ('SOUTH','EDSEL') is a BY group from the primary table, any matching rows from the Returns table are used to define the time series variables that are contributed from that table. The time series variables that are contributed by the Promotions table are always included in every BY group from the Sales table.

AUXDATA Alignment over the Time Dimension

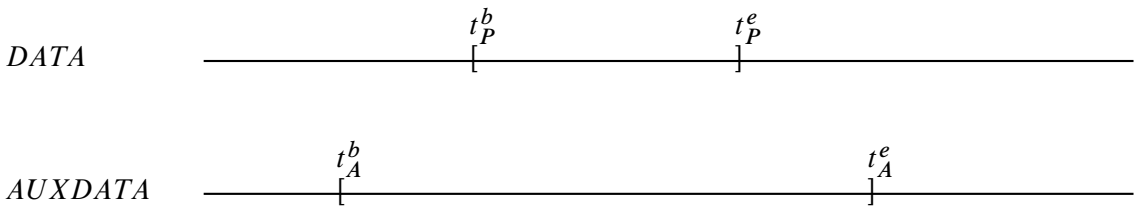
The series from each BY group of the primary table defines a reference time span for the auxiliary tables. Only the intersection of the time span for each auxiliary series with the reference span is input. Leading or trailing missing values are added to the auxiliary series as required to create a time series that has the same span as the reference span. The leading or trailing missing values, if any, are then interpreted according to the value of the SETMISSING= option. For more information, see the SETMISSING= option in the ID statement and the SETMISSING= option in the VAR statement.

When time series are input from a single primary table, the values for all observations of all the time series are contained in the primary table, and no time series needs to be extended with leading or trailing missing values. However, when time series are input from both the primary table and an auxiliary table, the time series from the auxiliary table are truncated or extended as required if the span of the auxiliary series is not identical to the reference span.

For the preceding (REGION, PRODUCT) example, which includes a primary table and two auxiliary tables, consider how differences in the reference span from each BY group affect the time series input from the auxiliary tables in the following cases. In these cases:

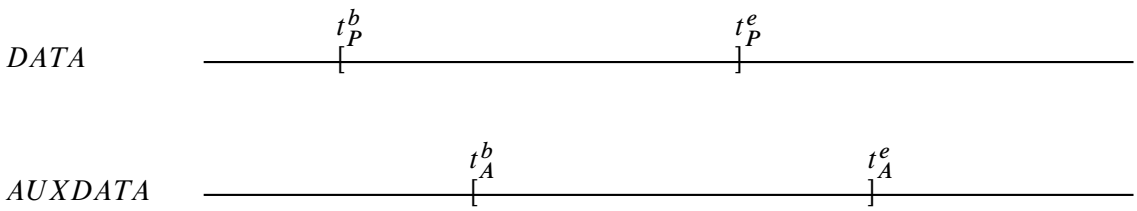
- The DATA line show a series that contains observations in the table specified in the DATA= option.
- The AUXDATA line show a series that contains observations in the table specified in an AUXDATA= option.
- t_P^b denotes the beginning time ID of the primary (DATA) series.
- t_P^e denotes the ending time ID of the primary (DATA) series.
- t_A^b denotes the beginning time ID of the AUXDATA series.
- t_A^e denotes the ending time ID of the AUXDATA series.
- $[t_P^b, t_P^e]$ denotes the time span for the primary (DATA) series (also known as the reference time span).
- $[t_A^b, t_A^e]$ denotes the time span for the AUXDATA series.

Case 1:



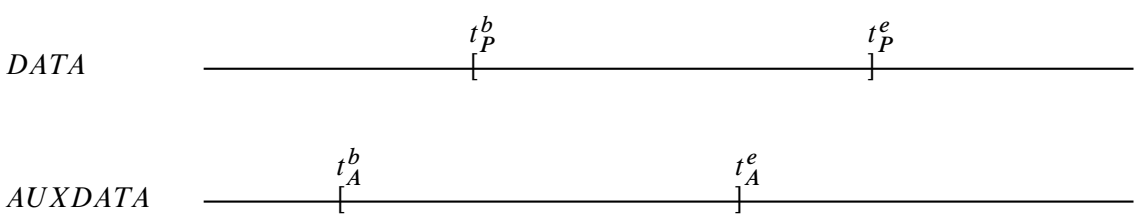
In this case, $[t_P^b, t_P^e] \subseteq [t_A^b, t_A^e]$: the auxiliary time span includes the reference span as a subset. Values in the AUXDATA series to the left of t_P^b and values to the right of t_P^e are truncated from the AUXDATA series. AUXDATA series values in $[t_P^b, t_P^e]$ are input as their actual values. Any actual missing values are interpreted according to the value of the SETMISSING= option.

Case 2:



In this case, $[t_P^b, t_P^e] = [t_P^b, t_A^b) \cup [t_A^b, t_P^e]$: the reference time span leads the auxiliary time span with a non-empty intersection. AUXDATA series values in $[t_P^b, t_A^b)$ are extended with missing values, and then those missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values in $[t_A^b, t_P^e]$ are input as their actual values. Any actual missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values in $(t_P^e, t_A^e]$ are truncated.

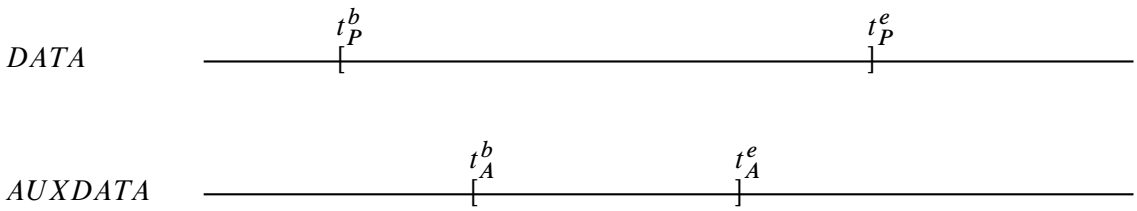
Case 3:



In this case, $[t_P^b, t_P^e] = [t_P^b, t_A^e] \cup (t_A^e, t_P^e]$: the reference time span lags the auxiliary time span with a non-empty intersection. AUXDATA series values in $[t_P^b, t_A^e]$ are input as their actual values. Any actual missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values

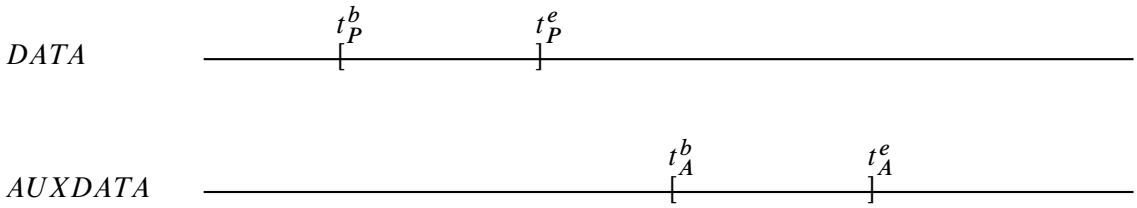
in $(t_A^e, t_P^e]$ are extended with missing values, and then those missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values in $(t_A^b, t_P^b]$ are truncated.

Case 4:



In this case, $[t_A^b, t_A^e] \subset [t_P^b, t_P^e]$: the auxiliary time span is a subset of the reference time span. AUXDATA series values in $[t_P^b, t_A^b)$ and values in $(t_A^e, t_P^e]$ are extended with missing values, and then those missing values are interpreted according to the value of the SETMISSING= option. AUXDATA series values in $[t_A^b, t_A^e]$ are input as their actual values. Any actual missing values are interpreted according to the value of the SETMISSING= option.

Case 5:



In this case, $[t_P^b, t_P^e] \cap [t_A^b, t_A^e] = \emptyset$: the auxiliary time span does not intersect the reference time span at all. AUXDATA series values in $[t_P^b, t_P^e]$ are set to missing values, and then those missing values are interpreted according to the value of the SETMISSING= option.

Table Output

The TSMODEL procedure can create the OUT=, OUTARRAY=, OUTSCALAR=, OUTSUM=, OUTLOG=, and OUTOBJ= tables. These tables always contain the variables that are specified in the BY statement. If a BY-group analysis step fails, then the values of this step are not recorded or are set to missing in the related output tables. Appropriate error or warning messages (or both) are recorded in the OUTLOG= table, subject to the value of the LOGCONTROL= option.

It is important to note that output tables created by PROC TSMODEL are naturally partitioned by the BY variables because of the way the BY groups are input into CAS session processes to form the time series data frames. The entire time series data for each BY group is input completely in a particular CAS session process. Consequently, all the time series processing for a particular BY group occurs in the context of a

single CAS session node. Subsequent use of PROC TSMODEL to process the output tables makes use of this table-level partitioning.

The BY group partitioning of the output tables should not be confused with BY-group sort order or time ID sort order when the output tables are read by the CAS libname engine back into the SAS DATA step code for subsequent processing. If BY-group order is needed, or if time ID sort order within the BY groups is needed, you must specify PROC SORT steps in your SAS code to order the data set. For example, consider the PROC TSMODEL step in the example "Comparison of the TSMODEL and TIMEDATA Procedures" on page 20. In order to process the OUTSUM=, OUTSCALAR=, and OUTARRAY= tables in a SAS DATA step that expects the BY-group rows to be contiguous, you must specify BY Product in the SORT procedure to sort the tables mycas.Pricexsum and mycas.Pricexos. The table mycas.Pricexoa is a time series table, and you must specify BY Product Date in the SORT procedure to sort the data so that they are suitable for normal SAS time series processing. Failure to sort output tables that are generated by PROC TSMODEL can lead to errors when the tables are used as input in other procedures or steps.

OUT= Table

The OUT= table contains the variables that are specified in the BY, ID, or VAR statements. The ID variable values are aligned and extended based on the ALIGN=, INTERVAL=, and LEAD= options. The values of the variables specified in the VAR statements are accumulated based on the ACCUMULATE= option, and missing values are interpreted based on the SETMISSING= option.

OUTARRAY= Table

The OUTARRAY= table contains the variables that are specified in the BY, ID or VAR statements. If the ID statement is specified, then the ID variable values are aligned and extended based on the ALIGN= and INTERVAL= options. The values of the variables specified in the VAR statements are accumulated based on the ACCUMULATE= option, and missing values are interpreted based on the SETMISSING= option. In addition, the OUTARRAY= table contains the variables that are specified in the OUTARRAYS statements and the following variables:

STATUS	status flag that indicates whether the requested analyses were successful
TIMEID	time ID values
SEASON	season index values
CYCLE	life-cycle index values
Array-Variable-Names	variables that are specified in the OUTARRAYS statement

The OUTARRAY= table contains the arrays that are related to the (accumulated) time series.

OUTSCALAR= Table

The OUTSCALAR= table contains the variables that are specified in the BY statement. In addition, the table contains the variables that are specified in the OUTSCALARS statements and the following variables:

STATUS	status flag that indicates whether the requested analyses were successful
Scalar-Variable-Names	variables that are specified in the OUTSCALARS statement

The OUTSCALAR= table contains the scalars that are related to the (accumulated) time series.

OUTSUM= Table

The OUTSUM= table contains the variables that are specified in the BY statement and the variables in the following list. This table also records the descriptive statistics for each variable that is specified in a VAR statement. Variables that are related to descriptive statistics are based on the ACCUMULATE= and SETMISSING= options in the ID and VAR statements.

NAME	variable name
STATUS	status flag that indicates whether the requested analyses were successful
START	the starting date of each series
END	the ending date of each series
STARTOBS	the beginning observation number of each series
ENDOBS	the ending observation number of each series
NOBS	number of observations
N	number of nonmissing observations
NMISS	number of missing observations
MINIMUM	minimum value
MAXIMUM	maximum value
AVG	average value
STDDEV	standard deviation

OUTLOG= Table

The OUTLOG= table contains the variables that are specified in the BY statement and the variables in the following list. The OUTLOG= table records textual messages that arise from the processing of the BY group's time series data. Messages are filtered based on the value of the LOGCONTROL= option. In addition to the BY variables, the OUTLOG= table includes the following columns:

ERRNO	a numeric variable that stores the _ERRNO_ variable for the BY group. The value of _ERRNO_ might be set by the user-defined program directly or might be set implicitly by calling a function or method that sets the _ERRNO_ value.
LOGLEN	a numeric variable that stores the length of the _LOG_ variable text (byte count).
LOG	a character variable that stores the messages that are logged from the execution of the user-defined program on the BY group's time series data. All messages from the BY group are concatenated into the variable. End-of-line characters separate the individual messages. If PUTTOLOG=YES is specified in the PROC TSMODEL statement, the _LOG_ variable also contains messages from any PUT programming statements that are specified.

STATUS Variable Values

The _STATUS_ variable that appears in the OUTARRAY=, OUTSCALAR= and OUTSUM= tables contains a value that specifies whether the analysis has been successful or not. The _STATUS_ variable can take the following values:

0	Analysis was successful.
3000	Accumulation failed.
4000	Missing value interpretation failed.
6000	Series is all missing.
9000	Descriptive statistics could not be computed.

Printed Output

The TSMODEL procedure always prints a summary of the processing that is performed on the time series data. This is extremely useful for gauging the work that is performed by the CAS server when it executes PROC TSMODEL. Printing of other results is best accomplished by the use of targeted data queries to subset and display the information from the tables that are produced by the TSMODEL procedure. For example, you might want to print the OUTARRAY= results only for BY groups that have a particular _STATUS_ value in the OUTSUM= table.

Examples: TSMODEL Procedure

Example 5.1: Accumulating Transactional Data into Time Series Data

This example uses the TSMODEL procedure to accumulate timestamped transactional data that have been recorded at no particular frequency into time series data at a specific frequency.

Suppose that the input table mycas.retail contains variables Store and Timestamp and numerous other numeric transaction variables. The BY variable Store contains values that divide the transactions into groups (BY groups). The time ID variable Timestamp contains SAS date values that are recorded at no particular frequency. The other variables, Item1–Item8, contain the numeric transaction values to be analyzed. The following statements form monthly time series from the transactional data. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
proc tsmodel data=mycas.retail
     out=mycas.mseries(replace=yes);
   by store;
   id timestamp interval=month
                accumulate=avg
                setmiss=0
                start='01jan1998'd
                end  ='31dec2000'd;
   var item1-item8;
run;
```

The ACCUMULATE=AVG option in the ID statement accumulates the transactions that are recorded with each time period based on the average values. The SETMISS=0 option in the ID statement sets the accumulated time series values for time periods that have no transactions to 0 instead of to missing. The START='01JAN1998'D and END='31DEC2000'D options request that only transactions recorded between the first day of 1998 and the last day of 2000 be considered and, if needed, extended to include this range.

The monthly time series data are stored in the table mycas.mseries. Each BY group that is associated with the BY variable Store contains an observation for each of the 36 months that are associated with the years 1998, 1999, and 2000. Each observation contains the values Store, Timestamp, and each of the analysis variables Item1–Item8 in the input table.

The TSMODEL procedure prints a summary of the time series processing that is performed, as shown in Output 5.1.1.

Output 5.1.1 Summary of Accumulation Processing

The TSMODEL Procedure

Summary of time series processing for RETAIL	
Number of analysis variables	8
Number of rows read	400
Number of groups read	4
Memory for group packages (KB)	40
Time to read groups (seconds)	0.0005481243
Time to load groups (seconds)	0.0040278435
Number of data threads	3
Thread BY group redundancy	1
Minimum time ID	NOV1998
Maximum time ID	DEC2000
Minimum time periods	36
Maximum time periods	36
Number of nodes run	1
Number of nodes with data	1
Number of nodes with groups	1
Number of threads budgeted	32
Minimum thread group count	1
Maximum thread group count	1
Minimum threads active	1
Maximum threads active	4
Average CPU utilization of nodes with data (%)	100
Minimum CPU utilization of nodes with data (%)	100
Maximum CPU utilization of nodes with data (%)	100
Number of groups processed by submitted code	4
Number of groups failing	0
Elapsed time to process groups (seconds)	0.0018050671
Number of output table rows produced	144

After each set of transactions has been accumulated to form a corresponding time series, the accumulated time series can be analyzed using various time series analysis techniques. For example, exponentially weighted moving averages can be used to smooth each series.

If the time ID variable Timestamp contains SAS datetime values instead of SAS date values, the INTERVAL=, START=, and END= options must be changed accordingly and the following statements could be used:

```
proc tsmodel data=mycas.retail
   out=mycas.tseries(replace=yes);
   by store;
   id timestamp interval=dtmonth
                accumulate=median
                setmiss=0
                start='01jan1998:00:00:00'dt
                end  ='31dec2000:00:00:00'dt;
   var item1-item8;
run;
```

The monthly time series data are stored in the table mycas.tseries, and the time ID values use a SAS datetime representation.

Example 5.2: User-Defined Program Statements

The following SAS macro statements create a user-defined subroutine and a user-defined function. Mylog is a subroutine that log-transforms a time series. Mymean is a function that computes the mean of a time series.

```
%macro fcmpcode;
   subroutine mylog(actual[*], transform[*]);
   outargs transform;
   actlen  = DIM(actual);
   do t = 1 to actlen;
      transform[t] = log(actual[t]);
   end;
   endsub;

   function mymean(z[*]);
   nz  = DIM(z);
   sum = 0;
   nnmiss = 0;
   do t = 1 to nz;
      if z[t] ne . then do;
         sum = sum + z[t];
         nnmiss = nnmiss + 1;
      end;
   end;
   if nnmiss eq 0 then return(.);

   return(sum/nnmiss);
   endsub;
%mend;
```

The following DATA step loads data set Sashelp.Air, which contains airline data into the table mycas.air. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.air(replace=yes);
    set sashelp.air;
run;
```

The input table mycas.air contains the variables Air and Date. The time series is recorded monthly.

The following statements form quarterly time series from the monthly series based on the median value of the total of the transactions recorded within each month. The OUTARRAYS statement specifies the Logair and Logairc arrays as output. The OUTSCALARS statement specifies the Meanlog scalars as output. The other arrays and scalars are not part of the output. The subsequent program statements create the output arrays and scalars.

```
proc tsmodel data=mycas.air
   outarray=mycas.arrays(replace=yes)
   outscalar=mycas.scalars(replace=yes)
   outsum=mycas.airsum(replace=yes);
   id date interval=qtr acc=total;
   var air;
   outarrays logair logairc;
   outscalars meanlog;
   submit;
      %fcmpcode;
      call mylog(air,logair);
      meanlog = mymean(air);
      do t = 1 to dim(air);
         logairc[t] = logair[t] - meanlog;
      end;
   endsubmit;
quit;
```

The following PROC PRINT step displays the OUTSUM= table, as shown in Output 5.2.1.

```
proc print data=mycas.airsum; run;
```

Output 5.2.1 Summary Statistics for mycas.Air

Obs	_NAME_	NOBS	N	NMISS	MIN	MAX	MEAN	STDDEV
1	AIR	48	48	0	341	1736	840.89583333	356.43144067
2	logair	48	48	0	5.8318824773	7.4593388952	6.6430769906	0.4395319391
3	logairc	48	48	0	-835.0639509	-833.4364944	-834.2527563	0.4395319394

Obs	_STATUS_	START	END	STARTOBS	ENDOBS
1	0	1949:1	1960:4	1	48
2	0	1949:1	1960:4	1	48
3	0	1949:1	1960:4	1	48

Example 5.3: Using Auxiliary Tables

This example demonstrates the use of the AUXDATA= option in the PROC TSMODEL statement. The data set Sashelp.Gulfoil contains oil and gas production data from the Gulf of Mexico. The following DATA step loads data set Sashelp.Gulfoil into the table mycas.gulfoil.

These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.gulfoil(replace=yes);
    set sashelp.gulfoil;
run;
```

The variable RegionName define a time series hierarchy of interest. Suppose you want to generate two new series that contain the region's total share of oil and gas production for each month in the mycas.gulfoil table.

You first use PROC TSMODEL to perform temporal aggregation (accumulation) of the time series:

```
proc tsmodel data=mycas.gulfoil
     outarray=mycas.allreg(replace=yes);
   id date interval=month accumulate=total;
   var oil gas;
   outarray alloil allgas;
   submit;
     do t=1 to _length_;
        alloil[t]=oil[t];
        allgas[t]=gas[t];
     end;
   endsubmit;
quit;
```

In the preceding PROC TSMODEL statements, the Oil and Gas variables are accumulated across all regions because no BY statement is specified. In the following DATA step, the variables are dropped from the mycas.allreg table to avoid conflict with the original time series variables in the mycas.gulfoil table:

```
data mycas.allreg;
    set mycas.allreg;
    drop oil gas;
run;
```

You can then use PROC TSMODEL with the AUXDATA= option to compute the share of oil and gas production that is contributed by each region for each month. PROC TSMODEL reads a monthly time series for each RegionName group for the variables Oil and Gas from Mycas.Gulfoil. Two new series are produced in the variables Oilshare and Gasshare, which contain the region's share of the oil and gas production, respectively. Those share variables are specified in the OUTARRAY statement for inclusion in the table (mycas.shares), which is specified in the OUTARRAY= option. The time series that are acquired for the variables alloil and allgas are common across all of the RegionName BY groups.

```
proc tsmodel data=mycas.gulfoil
             auxdata=mycas.allreg
             outarray=mycas.shares;
   by regionname;
   outarray oilshare gasshare;
   var oil gas alloil allgas;
   id date interval=month accumulate=total;
   submit;
      do i=1 to _length_;
         oilshare[i] = oil[i] / alloil[i];
         gasshare[i] = gas[i] / allgas[i];
      end;
   endsubmit;
quit;
```

The following code demonstrates that the computed shares sum to 1 for each time index in the resulting Oilshare and Gasshare series. PROC TSMODEL is used to accumulate the shares for these respective variables from the table mycas.shares, and the accumulated share series at the RegionName level are stored to the table mycas.rshares with variable names Oilsum and Gassum, respectively. The tables mycas.shares and mycas.rshares are shown in Output 5.3.1.

```
proc tsmodel data=mycas.shares
     outscalar=mycas.rshares(replace=yes)
     outsum=mycas.rsum(replace=yes);
   id date interval=month accumulate=total;
   var oilshare gasshare;
   outscalar oilsum gassum;
   submit;
      oilsum = 0;
      gassum = 0;
      do t=1 to _length_;
        oilsum = oilsum + oilshare[t];
        gassum = gassum + gasshare[t];
      end;
   endsubmit;
quit;

proc print data=mycas.rshares; run;
proc print data=mycas.rsum; run;
```

Output 5.3.1 Validation of Oil and Gas Shares by Region

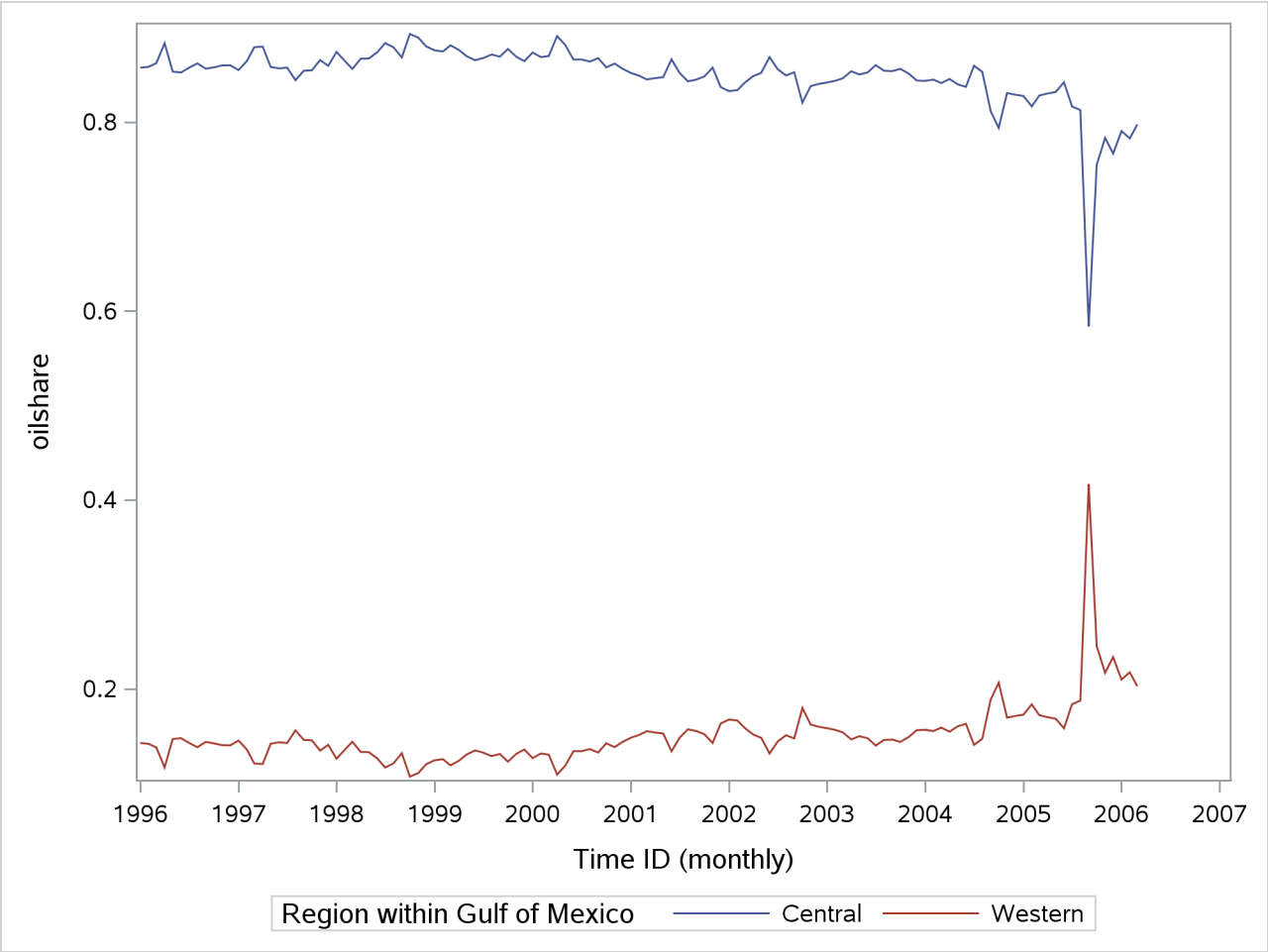
Obs	_STATUS_	oilsum	gassum
1	0	123	123

Obs	_NAME_	NOBS	N	NMISS	MIN	MAX	MEAN	STDDEV	_STATUS_	START	END	STARTOBS	ENDOBS
1	oilshare	123	123	0	1	1	1	0	0	JAN1996	MAR2006	1	123
2	gasshare	123	123	0	1	1	1	0	0	JAN1996	MAR2006	1	123

You might also want to plot the share series. The following code produces a graph that overlays the region level share series for oil production as shown in Output 5.3.2.

```
proc sgplot data=mycas.shares;
   series x=Date y=OilShare/group=RegionName;
run;
```

Output 5.3.2 Region Share of Oil Production



References

Keogh, E., Chu, S., Hart, D., and Pazzani, M. (2004). "Segmenting Time Series: A Survey and Novel Approach." In *Data Mining in Time Series Databases*, edited by M. Last, A. Kandel, and H. Bunke, 1–22. London: World Scientific. http://www.worldscientific.com/worldscibooks/10.1142/5210.

National Retail Federation (2007). "National Retail Federation 4-5-4 Calendar." http://www.nrf.com/.

Technical Committee ISO/TC 154 (Processes, Data Elements, and Documents in Commerce, Industry, and Administration) (2004). *ISO 8601:2004 Data Elements and Interchange Formats—Information Interchange—Representation of Dates and Times.* 3rd ed. Technical report, International Organization for Standardization.

Chapter 6
The TSRECONCILE Procedure

Contents

Overview: TSRECONCILE Procedure **61**
PROC TSRECONCILE Compared with the HPFRECONCILE procedure 62
Getting Started: TSRECONCILE Procedure **62**
Syntax: TSRECONCILE Procedure **64**
Functional Summary 64
PROC TSRECONCILE Statement 66
BY Statement 68
CHILDROLES Statement 68
ID Statement 69
PARENTROLES Statement 69
Details: TSRECONCILE Procedure **70**
Notation 70
Top-Down Reconciliation 71
Input and Output Data Tables 74
Examples: TSRECONCILE Procedure **75**
Example 6.1: Reconciling a Hierarchical Tree 75

Overview: TSRECONCILE Procedure

When data are organized in a hierarchical fashion, there are often accounting constraints that link the data at different levels of the hierarchy. Typically, for any particular time period, the data in a *parent* node are either the sum or the average of the data of its *child* nodes. For example, the total sales of a product by a retail company are the sum of the sales of the same product in all its stores.

It seems natural to require that the same constraints hold also for the predicted values. Imposing such constraints during the forecasting process can be difficult or impossible. Therefore, the series are often forecast independently at different levels, but this independence results in forecasts do not abide by the constraints that bind the original series. However, you can enforce these constraints by using an after-the-fact process that is called *reconciliation of hierarchical forecasts*.

The TSRECONCILE procedure reconciles forecasts of timestamped data at two different levels of a hierarchy in a top-down fashion for input data that are contained in CAS tables.

PROC TSRECONCILE Compared with the HPFRECONCILE procedure

The TSRECONCILE procedure is the next generation of the HPFRECONCILE procedure (in SAS Forecast Server Procedures) for hierarchical time series forecasts reconciliation. PROC TSRECONCILE was developed specifically for SAS Viya. The syntax of PROC TSRECONCILE is similar to the syntax of PROC HPFRECONCILE, from which it borrows the underlying methodology and goals. PROC TSRECONCILE is designed to run on a cluster of machines that distribute the data and computations in multiple threads, and it requires your data tables to be available on a SAS Cloud Analytic Server. Table 6.1 shows the differences between the TSRECONCILE and HPFRECONCILE procedures.

Table 6.1 PROC TSRECONCILE versus PROC HPFRECONCILE

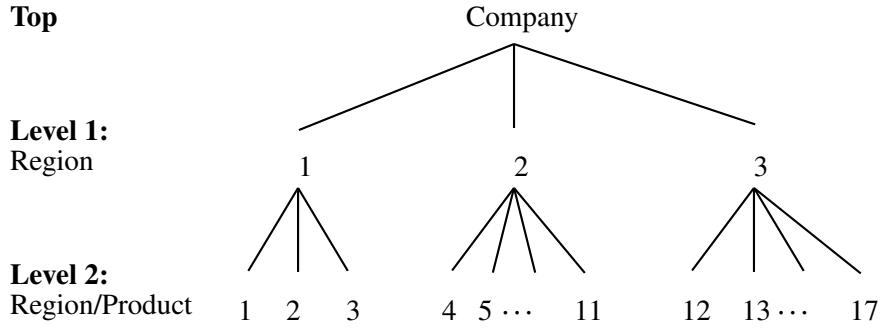
Feature	TSRECONCILE	HPFRECONCILE
Threading	Multithreaded	Single-threaded
Input data	CAS tables	SAS data sets
Requires sorted input data?	No	Yes, by BY variables and ID variable
Requires equally spaced data?	No, the INTERVAL= option in the ID statement is not supported	Yes, the INTERVAL= option in the ID statement indicates the frequency
Reconciliation direction supported	Top-down	Top-down and bottom-up

Getting Started: TSRECONCILE Procedure

This example uses the Sashelp.Pricedata data set to illustrate how to reconcile two levels of a hierarchy of forecasts in a top-down fashion.

The hierarchical structure of data set is shown in Figure 6.1.

Figure 6.1 Hierarchical Structure of Sashelp.Pricedata



The following DATA step loads the Pricedata data set from the Sashelp directory to a table named mycas.pricedata in your CAS session. This DATA step assumes that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
data mycas.pricedata(replace=yes);
   set sashelp.pricedata;
run;
```

The following statements use the TSMODEL procedure to generate forecasts for the dependent variable sale first at level 2 (region / product) and then at level 1 (region). For more information about the TSMODEL procedure, see Chapter 5, "The TSMODEL Procedure."

```
/* Define a macro to handle bad return codes of TSModel*/
%macro rccheck;
   if rc < 0 then do; stop; end;
%mend;

/* Automatically select model and generate forecasts at the child level */
proc tsmodel data=mycas.pricedata
       outobj=(outfor=mycas.outfor_region_product)
       errorstop=yes
       ;
   by region product;
   require atsm;
   id date interval=month;
   var sale;
   submit;
      declare object t(tsdf);
      declare object f(foreng);
      declare object outfor(outfor);
      rc = t.Initialize(); %rccheck;
      rc = t.AddY(sale); %rccheck;
      rc = f.Initialize(t); %rccheck;
      rc = f.SetOption('lead',12); %rccheck
      rc = f.Run(); %rccheck;
      rc = outfor.Collect(f); %rccheck;
   endsubmit;
run;

/* Automatically select model and generate forecasts at the parent level */
proc tsmodel data=mycas.pricedata
       outobj=(outfor=mycas.outfor_region)
       errorstop=yes
       ;
   by region;
   require atsm;
   id date interval=month;
   var sale;
   submit;
      declare object t(tsdf);
      declare object f(foreng);
      declare object outfor(outfor);
      rc = t.Initialize(); %rccheck;
      rc = t.AddY(sale); %rccheck;
      rc = f.Initialize(t); %rccheck;
      rc = f.SetOption('lead',12); %rccheck
      rc = f.Run(); %rccheck;
      rc = outfor.Collect(f); %rccheck;
   endsubmit;
run;
```

The following statements use the TSRECONCILE procedure to reconcile the separate forecasts in a top-down fashion:

```
/* Reconcile child to parent with default options */
proc tsreconcile
      child = mycas.outfor_region_product
      parent = mycas.outfor_region
      outfor = mycas.recfor_region_product
   ;
   by region product;
   id date;
run;
```

PROC TSRECONCILE requires the following inputs:

- The child table at the region / product level, specified in the CHILD= option in the PROC statement
- The parent table at the region level, specified in the PARENT= option in the PROC statement
- An output table, specified in the OUTFOR= option in the PROC statement
- BY variables at the child level (region / product), specified in the BY statement
- The variable that contains the timestamps in the ID statement

Syntax: TSRECONCILE Procedure

The following statements are available in the TSRECONCILE procedure:

PROC TSRECONCILE < *options* > **;**
  BY *variables* **;**
  ID *variable* < *options* > **;**
  CHILDROLES < *options* > **;**
  PARENTROLES < *options* > **;**

The following sections describe the PROC TSRECONCILE statement and then describe the other statements in alphabetical order.

Functional Summary

Table 6.2 summarizes statements and options that are used with the TSRECONCILE procedure.

Table 6.2 PROC TSRECONCILE Functional Summary

Description	Statement	Option
Statements		
Specifies the BY variables	BY	
Specifies the time ID variable	ID	
name variables in the CHILD= data table	CHILDROLES	
name variables in the PARENT= data table	PARENTROLES	
Time ID Options		
Specifies the ending time ID value	ID	END=
Specifies the starting time ID value	ID	START=
Data Table Options		
Specifies the child-level input data table	PROC TSRECONCILE	CHILD=
Specifies the parent-level input data table	PROC TSRECONCILE	PARENT=
Specifies the output data table to contain the reconciled forecasts	PROC TSRECONCILE	OUTFOR=
Specifies the variable that contains the actual values in the CHILD= data table	CHILDROLES	ACTUAL=
Specifies the variable that contains the actual values in the PARENT= data table	PARENTROLES	ACTUAL=
Specifies the variable that contains the predicted values in the CHILD= data table	CHILDROLES	PREDICT=
Specifies the variable that contains the predicted values in the PARENT= data table	PARENTROLES	PREDICT=
Specifies the variable that contains the lower confidence limit in the CHILD= data table	CHILDROLES	LOWER=
Specifies the variable that contains the lower confidence limit in the PARENT= data table	PARENTROLES	LOWER=
Specifies the variable that contains the upper confidence limit in the CHILD= data table	CHILDROLES	UPPER=
Specifies the variable that contains the upper confidence limit in the PARENTROLES= data table	PARENTROLES	UPPER=
Specifies the variable that contains the prediction error in the CHILD= data table	CHILDROLES	ERROR=
Specifies the variable that contains the prediction error in the PARENT= data table	PARENTROLES	ERROR=
Specifies the variable that contains the standard error in the CHILD= data table	CHILDROLES	STD=
Specifies the variable that contains the standard error in the PARENT= data table	PARENTROLES	STD=

Description	Statement	Option
Analysis Options		
Specifies the aggregation method	PROC TSRECONCILE	AGGREGATE=
Specifies the confidence level	PROC TSRECONCILE	ALPHA=
Specifies method for confidence limits	PROC TSRECONCILE	CLMETHOD=
Specifies the disaggregation method	PROC TSRECONCILE	DISAGGREGATION=
Leaves zeros forecast unchanged	PROC TSRECONCILE	LOCKZERO
Specifies sign bound on the reconciled series	PROC TSRECONCILE	SIGN=
Requests that the loss function be weighted by the inverse of the prediction variances	PROC TSRECONCILE	WEIGHTED

PROC TSRECONCILE Statement

PROC TSRECONCILE *options* **;**

The PROC TSRECONCILE statement invokes the TSRECONCILE procedure. You can specify the *options* that are described in the following subsections.

Required Options Related to Input Data Tables

You must specify the following data tables:

CHILD=*CAS-libref.data-table*

DISDATA=*CAS-libref.data-table*

DATA=*CAS-libref.data-table*

specifies the input data table that contains the forecast of the time series that represent the children nodes in the hierarchy. Typically, the CHILD= data table is generated by an OUTFOR= data collector object of the TSMODEL procedure. The dimensions of the CHILD= data table are greater than the dimensions of the PARENT= data table. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table.

For more information, see the section "CHILD= Data Table" on page 74.

PARENT =*CAS-libref.data-table*

AGGDATA=*CAS-libref.data-table*

specifies the input data table that contains the forecasts of the time series that represent the parent nodes in the hierarchy. Typically, the PARENT= data table is generated by an OUTFOR= data collector object of the TSMODEL procedure. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table.

The PARENT= data table must contain a proper subset, possibly empty, of the BY variables that are present in the CHILD= data table.

For more information, see the section "PARENT= Data Table" on page 74.

Required Option Related to the Output Data Table

You must specify the following output data table:

OUTFOR=*CAS-libref.data-table*
specifies the name of the output data table in which to store the reconciled values. *CAS-libref.data-table* is a two-level name, where *CAS-libref* refers to the caslib and session identifier, and *data-table* specifies the name of the input data table.

For more information, see the section "OUTFOR= Data Table" on page 74.

Options Related to the Analysis

AGGREGATE=SUM | AVERAGE
specifies whether the dependent variable in the PARENT= data table is the total sum or the average over the BY groups of the dependent variable in the CHILD= data table. By default, AGGREGATE=SUM.

ALPHA=α
specifies the level of the confidence limits when CLMETHOD=GAUSSIAN, where α must be between 0.0001 and 0.9999. The upper and lower confidence limits will have a $1-\alpha$ confidence level. By default, ALPHA=0.05, which produces 95% confidence intervals.

CLMETHOD=SHIFT | GAUSSIAN
specifies which method to use to compute confidence limits for the reconciled forecasts.

You can specify the following methods:

GAUSSIAN computes the confidence intervals by assuming that the forecasts are approximately Gaussian.

SHIFT computes the confidence intervals by re-centering the original confidence intervals around the new forecasts.

By default, CLMETHOD=SHIFT.

For more information about the methods of computing confidence intervals, see the section "Details: TSRECONCILE Procedure" on page 70.

DISAGGREGATION=*option*
LOSS=*option*
specifies the type of disaggregation method and type of loss function for top-down reconciliation.

You can specify the following values:

DIFFERENCE bases the loss function on the root mean squared error (RMSE), which results in adjustments that are the (possibly weighted) mean difference of the aggregated child nodes and the parent node.

PROPORTIONS uses a loss function that results in reconciled forecasts that are the (possibly weighted) proportional disaggregation of the parent node.

For more information about the disaggregation methods and associated loss functions, see the section "Top-Down Reconciliation" on page 71.

By default, DISAGGREGATION=PROPORTIONS.

LOCKZERO
specifies that a value of 0 for the Predict variable in the CHILD= data table implies a value of 0 for the corresponding observation in the OUTFOR= data table.

SIGN=*option*
specifies the sign in the reconciled series.

You can specify the following values:

MIXED | NONE	specifies that the output series can have any sign.
NONNEGATIVE | POSITIVE	specify that the output series is nonnegative.
NONPOSITIVE | NEGATIVE	specifies that the output series is nonpositive.

By default, SIGN=MIXED.

WEIGHTED
weights the loss function for top-down reconciliation by the inverse of the variance of the input forecasts.

BY Statement

BY *variables* **;**

The BY statement defines separate groups of observations for the CHILD= data table. BY variables can be either character or numeric.

All BY variables must exist in the CHILD= data table. Conversely, only a strict subset of or none of the BY variables must be present in the PARENT= data table. The BY variables that are present in the PARENT= data table are called AGGBY or PARENTBY variables. Because the AGGBY variables form a proper subset of the BY variables, their number must be less than the number of BY variables. PROC TSRECONCILE finds the AGGBY variables by comparing the variables in the BY statement with the variables in the PARENT= data table.

CHILDROLES Statement

CHILDROLES < *options* > **;**

The CHILDROLES statement enables you to specify names for forecasting variables in the CHILD= data table.

You can specify the following *options*:

ACTUAL=*variable-name*
specifies the name of the variable in theCHILD= data table that contains the actual values. By default, ACTUAL=ACTUAL.

PREDICT=*variable-name*
specifies the name of the variable in the CHILD= data table that contains the predicted values. By default, PREDICT=PREDICT.

LOWER=*variable-name*
specifies the name of the variable in the CHILD= data table that contains the lower confidence limit values. By default, LOWER=LOWER.

UPPER=*variable-name*
specifies the name of the variable in the CHILD= data table that contains the upper confidence limit values. By default, UPPER=UPPER.

ERROR=*variable-name*
specifies the name of the variable in the CHILD= data table that contains the error values. By default, ERROR=ERROR.

STD=*variable-name*
specifies the name of the variable in the CHILD= data table that contains the standard error values. By default, STD=STD.

ID Statement

ID *variable* < /*options* > **;**

The ID statement names a numeric variable that identifies observations in the input and output data tables. The ID variable's values are assumed to be SAS date, time, or datetime values.

You can specify the following *options* in the ID statement:

END=*option*
specifies a SAS date, datetime, or time value that represents the date at which the reconciliation ends. If the largest *variable* value is less than the END= value, this option has no effect.

START=*option*
specifies a SAS date, datetime, or time value that represents the *variable* value at which the reconciliation begins. This option can be used to limit the reconciliation process only to forecasts that are outside the historical period.

PARENTROLES Statement

PARENTROLES < *options* > **;**

The PARENTROLES statement enables you to specify custom names for forecasting variables in the PARENT= data table.

You can specify the following *options*:

ACTUAL=*variable-name*
specifies the name of the variable in the PARENT= data table that contains the actual values. By default, ACTUAL=ACTUAL.

PREDICT=*variable-name*
specifies the name of the variable in the PARENT= data table that contains the predicted values. By default, PREDICT=PREDICT.

LOWER=*variable-name*
specifies the name of the variable in the PARENT= data table that contains the lower confidence limit values. By default, LOWER=LOWER.

UPPER=*variable-name*
specifies the name of the variable in the PARENT= data table that contains the upper confidence limit values. By default, UPPER=UPPER.

ERROR=*variable-name*
specifies the name of the variable in the PARENT= data table that contains the error values. By default, ERROR=ERROR.

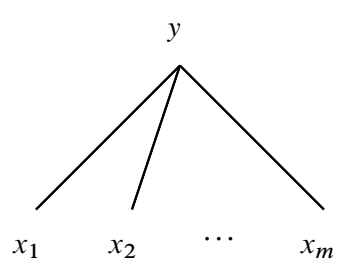
STD=*variable-name*
specifies the name of the variable in the PARENT= data table that contains the standard error values. By default, STD=STD.

Details: TSRECONCILE Procedure

Notation

Assume a two-level hierarchical structure as depicted in Figure 6.2, where y is a parent series and x_i, $i = 1 \ldots m$ are the child series.

Figure 6.2 Hierarchical Structure



Let y_t be the values of the parent series at time t, and let $\mathbf{x}_t = [x_{1,t}, x_{2,t}, \ldots, x_{m,t}]'$ be the vector of child series at time t, $t = 1, \ldots, T$. As usual, indicate by $\hat{y}_t$ and $\hat{\mathbf{x}}_t$ the prereconciliation forecasts of y_t and $\mathbf{x}_t$, respectively, and denote by $\hat{\boldsymbol{\sigma}}_t = [\hat{\sigma}_{1,t}, \hat{\sigma}_{2,t}, \ldots, \hat{\sigma}_{m,t}]'$ the vector of prediction standard error for $\hat{\mathbf{x}}_t$. Denote by $\hat{\boldsymbol{\Sigma}}$ the diagonal matrix whose main diagonal is $\hat{\boldsymbol{\sigma}}_t^2$. Let a tilde indicate the reconciled values, so that $\tilde{y}_t$

and $\tilde{\mathbf{x}}_t$ indicate the reconciled values of $\hat{y}_t$ and $\hat{\mathbf{x}}_t$, respectively. The number of child series m can vary with t; however, for simplicity, the number of child series is considered fixed in the following discussion.

At each time t, the values of the series $x_{i,t}$, $i = 1 \ldots, m$, and y_t are bound by an aggregation constraint. For example, if the x_i are the sales at store level for a retail company, then y_t can be either the total sales at company level or the average sales per store. If you specify the AGGREGATE=SUM option in the PROC TSRECONCILE statement, the aggregation constraint is $y_t = \sum_{i=1}^{m} x_{i,t}$. If instead you specify the AGGREGATE=AVERAGE option, the constraint is $y_t = \frac{1}{m} \sum_{i=1}^{m} x_{i,t}$.

If you need to have forecasts at both levels of the hierarchy, it is often more convenient to produce statistical forecasts separately for each series. However, the resulting forecasts do not abide by the aggregation constraint that binds the original series. The after-the-fact process through which the statistical forecasts are modified to enforce the aggregation constraint is called *reconciliation*.

By determining whether the upper-level forecasts or the lower-level forecasts are adjusted to meet the aggregation constraint, you can distinguish between bottom-up (BU) and top-down (TD) reconciliation, respectively. The TSRECONCILE procedure performs top-down reconciliation between two levels of a hierarchy of forecasts.

Top-Down Reconciliation

The goal of top-down (TD) reconciliation is to adjust the statistical forecasts $\hat{x}_{i,t}$ to obtain a new series $\{\tilde{x}_{i,t}\}$ of reconciled forecasts so that the sum of the reconciled forecasts at each fixed time t is equal to $\hat{y}_t$.

The problem can be restated as follows: minimize with respect to $\tilde{\mathbf{x}}$ a quadratic loss function $L(\tilde{\mathbf{x}}_t; \hat{\mathbf{x}}_t)$ subject to the following constraints.

- The *top-down constraint* is

$$\sum_{i=1}^{m} \tilde{x}_{i,t} = \hat{y}_t$$

- The lower bounds are

$$\tilde{x}_{i,t} \geq l_{i,t} \qquad i \in L_t \qquad L_t \in \{1, 2, \ldots, m\}$$

- The upper bounds are

$$\tilde{x}_{i,t} \leq u_{i,t} \qquad i \in U_t \qquad U_t \in \{1, 2, \ldots, m\}$$

Bounds are set by the SIGN= option in the TSRECONCILE statement, which specifies nonpositive or nonnegative bounds on all reconciled forecasts.

PROC TSRECONCILE uses an iterative interior point algorithm to solve the constrained quadratic optimization problem. For more information about this algorithm, see Chapter 17, "The Quadratic Programming Solver" (*SAS Optimization: Mathematical Optimization Procedures*).

Choice of Loss Function

The loss function takes either of the following functional forms, where $\mathbf{W}$ is a positive semidefinite matrix of weights independent of $\tilde{\mathbf{x}}_t$, $\hat{\mathbf{X}}^{-\frac{1}{2}}$ is a diagonal matrix with the square root of $\hat{\mathbf{x}}_t$ on the main diagonal, and $\overline{\hat{\mathbf{X}}^{-\frac{1}{2}}}$ is its complex conjugate:

- When DISAGGREGATION=DIFFERENCE, the loss function is defined as the following:

$$L(\tilde{\mathbf{x}}_t; \hat{\mathbf{x}}_t) = (\tilde{\mathbf{x}}_t - \hat{\mathbf{x}}_t)'\mathbf{W}^{-1}(\tilde{\mathbf{x}}_t - \hat{\mathbf{x}}_t)$$

- When DISAGGREGATION=PROPORTIONS, the loss function is defined as the following only when all $\hat{x}_{i,t}$ are different from 0. The solutions can be extended to the zero cases by defining $\tilde{x}_{i,t} := 0$ when $\hat{x}_{i,t} = 0$ if at least one $\hat{x}_{j,t}$ is different from 0. The case where all $\hat{x}_{j,t}$ are 0 is handled by setting $\tilde{x}_{i,t} := \frac{\hat{y}_t}{m}$ when AGGREGATE=TOTAL and $\tilde{x}_{i,t} := \hat{y}_t$ when AGGREGATE=AVERAGE.

$$L(\tilde{\mathbf{x}}_t; \hat{\mathbf{x}}_t) = (\tilde{\mathbf{x}}_t - \hat{\mathbf{x}}_t)'\hat{\mathbf{X}}^{-\frac{1}{2}}\mathbf{W}^{-1}\overline{\hat{\mathbf{X}}^{-\frac{1}{2}}}(\tilde{\mathbf{x}}_t - \hat{\mathbf{x}}_t)$$

If the WEIGHTED option is not specified in the PROC TSRECONCILE statement, $\mathbf{W}$ is the identity matrix $\mathbf{I}$. If the WEIGHTED option is specified, $\mathbf{W} = \hat{\mathbf{\Sigma}}$, the diagonal matrix with the estimated variances $\hat{\sigma}^2_{i,t}$ of $\hat{x}_{i,t}$ on the main diagonal. The standard errors must be strictly positive.

Unconstrained Solutions

When the only constraint is the top-down constraint, no bounds are present, and the WEIGHTED option is not specified (that is, $\mathbf{W} = \mathbf{I}$), the top-down problem admits intuitive solutions.

When DISAGGREGATION=DIFFERENCE, the loss function becomes

$$L(\tilde{\mathbf{x}}_t; \hat{\mathbf{x}}_t) = \sum_{i=1}^{m}(\hat{x}_{i,t} - \tilde{x}_{i,t})^2$$

This leads to the following solution, where $\hat{r}_t$ is the forecasting aggregate error:

$$\tilde{x}_{i,t} = \hat{x}_{i,t} + \frac{\hat{r}_t}{m}$$

When AGGREGATE=TOTAL,

$$\hat{r}_t := \hat{y}_t - \sum_{i=1}^{m}\hat{x}_{i,t}$$

When AGGREGATE=AVERAGE,

$$\hat{r}_t := m\hat{y}_t - \sum_{i=1}^{m}\hat{x}_{i,t}$$

Thus, when DISAGGREGATION=DIFFERENCE, the reconciled forecast $\tilde{x}_{i,t}$ is found by splitting the aggregation error $\hat{r}_t$ equally among the lower-level forecasts $\hat{x}_{i,t}$.

Notice that even if all statistical forecasts $\hat{x}_{i,t}$ are strictly positive, the reconciled forecasts $\tilde{x}_{i,t}$ are not necessarily positive if no bounds are specified in the SIGN= option. In particular, $\hat{x}_{i,t} = 0$ does not imply $\tilde{x}_{i,t} = 0$.

If DISAGGREGATION=PROPORTIONS, the loss function becomes

$$L(\tilde{\mathbf{x}}_t; \hat{\mathbf{x}}_t) = \sum_{i=1}^{m} \frac{\left(\hat{x}_{i,t} - \tilde{x}_{i,t}\right)^2}{|\hat{x}_{i,t}|}$$

This leads to the following solutions:

$$\tilde{x}_{i,t} = \hat{x}_{i,t} + \frac{|\hat{x}_{i,t}|}{\sum_{j=1}^{m} |\hat{x}_{j,t}|} \hat{r}_t$$

When AGGREGATE=TOTAL and all the $\hat{x}_{j,t}$ have the same sign, the solution resolves to

$$\tilde{x}_{i,t} = \frac{\hat{x}_{i,t}}{\sum_{j=1}^{m} \hat{x}_{j,t}} \hat{y}_t$$

When AGGREGATE=AVERAGE and all the $\hat{x}_{j,t}$ have the same sign, the solution resolves to

$$\tilde{x}_{i,t} = \frac{\hat{x}_{i,t}}{\sum_{j=1}^{m} \hat{x}_{j,t}} m \hat{y}_t$$

Thus, the reconciled forecast $\tilde{x}_{i,t}$ is found by disaggregating the upper-level forecasts according to the proportion that $\hat{x}_{i,t}$ represents in the total sum of the lower-level forecasts.

Missing Values

When one or more of the predicted values are missing, the missing values are replaced by the corresponding actual values that are present. This replacement is done in order to prevent bias between the aggregated and reconciled forecasts, which would result from models in which missing values in the predictions are generated because of the presence of lagged variables. If the corresponding actual value is also missing, the series is excluded from the reconciliation process.

When you use the WEIGHTED option and the standard error is missing, the weight is assumed to be equal to 1.

Confidence Limits

When CLMETHOD=SHIFT, the reconciled confidence limits are computed by re-centering the original confidence limits around the reconciled predicted values.

When CLMETHOD=GAUSS, the reconciled confidence limits are computed by assuming that the series is Gaussian with standard error equal to the prediction standard error.

Input and Output Data Tables

CHILD= Data Table

The CHILD= table must contain the variables that are specified in the BY statement, the variable that is specified in the ID statement, and the PREDICT variable, which specifies the predicted values.

The following variables can optionally be present in the CHILD= data table and are used when available. If not present, their value is assumed to be missing for computational purposes. If these variables are present in the CHILD= data table, they will be present also in the OUTFOR= data table. If the _NAME_ variable is present in both the CHILD= and PARENT= data tables, it is automatically used as a BY variable.

NAME	variable name
ACTUAL	actual values
LOWER	lower confidence limits
UPPER	upper confidence limits
ERROR	prediction errors
STD	prediction standard errors

Typically, the CHILD= data table is generated by an OUTFOR= data collector object of the TSMODEL procedure.

You can specify custom names for the variables in the CHILD= data table by using the CHILDROLES statement. For more information, see the section "CHILDROLES Statement" on page 68.

PARENT= Data Table

The PARENT= table must contain a proper subset (possibly empty) of the variables that are specified in the BY statement, the variable specified in the ID statement, and the PREDICT variable, which specifies the predicted values.

Typically, the PARENT= data table is generated by an OUTFOR= data collector object of the TSMODEL procedure.

You can specify custom names for the variables in the PARENT= table by using the PARENTROLES statement. For more information, see the section "PARENTROLES Statement" on page 69.

OUTFOR= Data Table

The OUTFOR= table contains the BY variables that are specified in the BY statement, the variable that is specified in the ID statement, the ERROR variable, which specifies the prediction errors, and the following variables if they are present in the CHILD= data table:

NAME	variable name
ACTUAL	actual values
PREDICT	predicted values
LOWER	lower confidence limits

UPPER	upper confidence limits
STD	prediction standard errors

Examples: TSRECONCILE Procedure

Example 6.1: Reconciling a Hierarchical Tree

The TSRECONCILE procedure reconciles forecasts between two levels of a hierarchy. It can also be used recursively for reconciling the entire hierarchy.

This example continues the example in the section "Getting Started: TSRECONCILE Procedure" on page 62. Consider the hierarchy structure for the Sashelp.Pricedata data set outlined in Figure 6.1. You can reconcile the hierarchy top down, starting from the top level 0 down to the bottom level 2. At each new iteration, the OUTFOR= data table of the previous reconciliation step becomes the PARENT= data table of the current step.

First, you need to compute the statistical forecasts for all levels. The statistical forecasts for level 1 and level 2 were already computed in the section "Getting Started: TSRECONCILE Procedure" on page 62, so only the forecasts at the company levels are left to compute in the following statements. These statements assume that your CAS engine libref is named mycas, but you can substitute any appropriately defined CAS engine libref.

```
 /* Forecast series at company level */
proc tsmodel data=mycas.pricedata
       outobj=(outfor=mycas.outfor_company)
       errorstop=yes
       ;
require atsm;
id date interval=month;
var sale;
submit;
  declare object t(tsdf);
  declare object f(foreng);
  declare object outfor(outfor);
  rc = t.Initialize(); %rccheck;
  rc = t.AddY(sale); %rccheck;
  rc = f.Initialize(t); %rccheck;
  rc = f.SetOption('lead',12); %rccheck
  rc = f.Run(); %rccheck;
  rc = outfor.Collect(f); %rccheck;
endsubmit;
run;
```

First, you reconcile the top and region levels. The output data table recfor_region (from the example in the section "Getting Started: TSRECONCILE Procedure" on page 62) contains the reconciled forecasts at level 1. This data table becomes the PARENT= data table for the next step of reconciliation, which involves level 1 and level 2.

```
 /* Reconcile forecasts top down from company to region */
proc tsreconcile
      parentTable = mycas.outfor_company
      childTable = mycas.outfor_region
      outfor = mycas.recfor_region
      ;
   by region;
   id date;
run;

/* Reconcile forecasts top down from region to region/product */
proc tsreconcile
      parentTable = mycas.recfor_region
      childTable = mycas.outfor_region_product
      outfor = mycas.recfor_region_product
      ;
   by region product;
   id date;
run;
```

The output data tables mycas.recfor_region and mycas.recfor_region_product contain the reconcile forecasts for the hierarchy at levels 1 and 2, respectively.

Subject Index

BY groups
    TSINFO procedure, 14
    TSMODEL procedure, 32
    TSRECONCILE procedure, 68

confidence limits
    top-down reconciliation (TSRECONCILE), 73

inferring time intervals and alignments
    TSINFO procedure, 15

missing values
    top-town reconciliation (TSRECONCILE), 73

time series information, *see also* TSINFO procedure
time series reconciliation, *see also* TSRECONCILE procedure
top-down reconciliation
    TSRECONCILE procedure, 71
TSINFO procedure
    BY groups, 14
TSMODEL procedure
    BY groups, 32
    input data tables, 28
    time series analysis, 19
TSRECONCILE procedure
    BY groups, 68

Syntax Index

ACCUMULATE= option
    ID statement (TSMODEL), 36
    VAR statement (TSMODEL), 41
AGGREGATE= option
    PROC TSRECONCILE statement, 67
ALIGN= option
    ID statement (TSINFO), 14
    ID statement (TSMODEL), 37
ALPHA= option
    PROC TSRECONCILE statement, 67
AUXDATA= option
    PROC TSMODEL statement, 28

BY statement
    TSINFO procedure, 14
    TSMODEL procedure, 32
    TSRECONCILE procedure, 68

CHILD= option
    PROC TSRECONCILE statement, 66
CHILDROLES statement
    TSRECONCILE procedure, 68
CLMETHOD= option
    PROC TSRECONCILE statement, 67

DATA= option
    PROC TSINFO statement, 14
    PROC TSMODEL statement, 28
DISAGGREGATION=option
    PROC TSRECONCILE statement, 67

END= option
    ID statement (TSMODEL), 37
    ID statement (TSRECONCILE), 69
ENDSUBMIT statement
    TSMODEL procedure, 43

FORMAT= option
    ID statement (TSMODEL), 38

ID statement
    TSINFO procedure, 14
    TSMODEL procedure, 33
    TSRECONCILE procedure, 69
INOBJ= option
    PROC TSMODEL statement, 29
INSCALAR= option
    PROC TSMODEL statement, 29
INSCALARS statement
    TSMODEL procedure, 39
INTERVAL= option
    ID statement (TSMODEL), 33

LEAD= option
    PROC TSMODEL statement, 30
LOCKZERO option
    PROC TSRECONCILE statement, 68
LOGCONTROL= option
    PROC TSMODEL statement, 30

NTHREADS= option
    PROC TSINFO statement, 14

OUT= option
    PROC TSMODEL statement, 30
OUTARRAY statement
    TSMODEL procedure, 40
OUTARRAY= option
    PROC TSMODEL statement, 31
OUTARRAYS statement
    TSMODEL procedure, 40
OUTFOR= option
    PROC TSRECONCILE statement, 67
OUTINTERVALDETAILS= option
    PROC TSINFO statement, 14
OUTLOG= option
    PROC TSMODEL statement, 31
OUTOBJ= option
    PROC TSMODEL statement, 31
OUTSCALAR statement
    TSMODEL procedure, 40
OUTSCALAR= option
    PROC TSMODEL statement, 31
OUTSCALARS statement
    TSMODEL procedure, 40
OUTSUM= option
    PROC TSMODEL statement, 32

PARENT=option
    PROC TSRECONCILE statement, 66
PARENTROLES statement
    TSRECONCILE procedure, 69
PRINT statement
    TSMODEL procedure, 40
PROC TSINFO statement, 13
PROC TSMODEL statement, 28
PROC TSRECONCILE statement, 66
programming statements

TSMODEL procedure, 43
PUTTOLOG= option
PROC TSMODEL statement, 32

REQUIRE statement
TSMODEL procedure, 41

SEASONALITY= option
PROC TSMODEL statement, 32
SETMISSING= option
ID statement (TSMODEL), 38
VAR statement (TSMODEL), 41
SIGN= option
PROC TSRECONCILE statement, 68
START= option
ID statement (TSMODEL), 38
ID statement (TSRECONCILE), 69
SUBMIT statement
TSMODEL procedure, 42

TRIMID= option
ID statement (TSMODEL), 39
TSINFO procedure
syntax, 13
TSINFO procedure, PROC TSINFO statement
DATA= option, 14
NTHREADS= option, 14
OUTINTERVALDETAILS= option, 14
TSMODEL procedure, 26, 28
syntax, 26, 28
TSMODEL procedure, PROC TSMODEL statement
DATA= option, 28
TSRECONCILE procedure
syntax, 64
TSRECONCILE procedure, PROC TSRECONCILE statement
AGGREGATE= option, 67
ALPHA= option, 67
CHILD= option, 66
CLMETHOD= option, 67
DISAGGREGATION=option, 67
LOCKZERO option, 68
OUTFOR= option, 67
PARENT= option, 66
SIGN= option, 68
WEIGHTED option, 68

VAR statement
TSMODEL procedure, 41

WEIGHTED option
PROC TSRECONCILE statement, 68