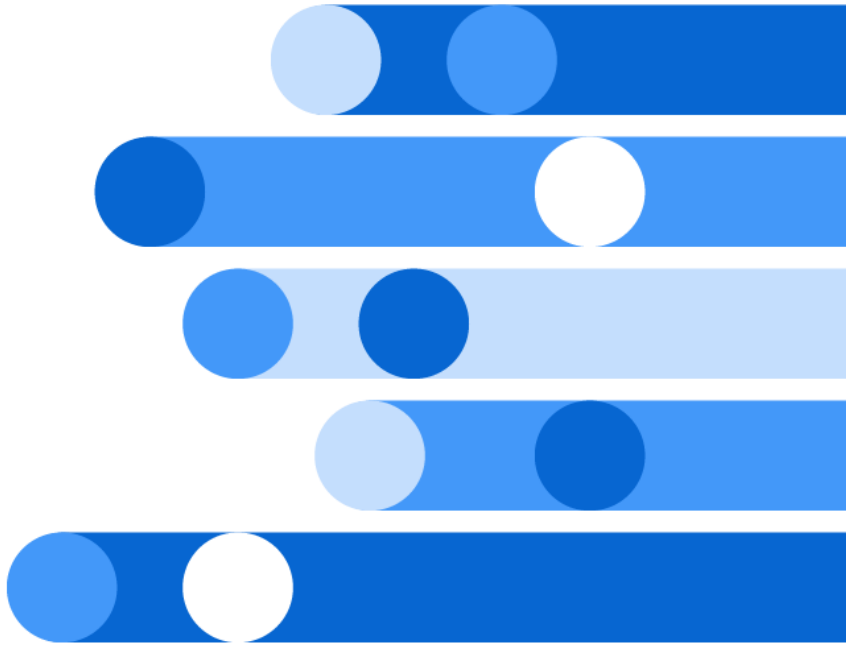
SAS/ETS® User's Guide

Date Intervals, Formats, and Functions

2025.01 - 2025.07*



* This document might apply to additional versions of the software. Open this document in SAS Help Center and click on the version in the banner to see all available versions.

SAS® Documentation
July 17, 2025

This document is an individual chapter from *SAS/ETS® User's Guide*.

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2025. *SAS/ETS® User's Guide*. Cary, NC: SAS Institute Inc.

SAS/ETS® User's Guide

July 2025

Chapter 4
Date Intervals, Formats, and Functions

Contents

Overview

This chapter summarizes the time intervals, date and datetime informats, date and datetime formats, and date, time, and datetime functions available in SAS software. The use of these features is explained in Chapter 3, "Working with Time Series Data." The material in this chapter is also contained in *SAS Programmers Guide: Essentials* and *Base SAS Procedures Guide*. Because these features are useful for work with time series data, documentation of these features is consolidated and repeated here for easy reference.

Time Intervals

This section provides a reference for the different kinds of time intervals supported by SAS software, but it does not cover how they are used. For an introduction to the use of time intervals, see Chapter 3, "Working with Time Series Data."

Some interval names are used with SAS date values, while other interval names are used with SAS date-time values. The interval names used with SAS date values are YEAR, SEMIYEAR, QTR, MONTH, SEMIMONTH, TENDAY, WEEK, WEEKDAY, DAY, YEARV, R445YR, R454YR, R544YR, R445QTR, R454QTR, R544QTR, R445MON, R454MON, R544MON, and WEEKV. The interval names used with SAS datetime or time values are HOUR, MINUTE, and SECOND. Various abbreviations of these names are also allowed, as described in the section "Summary of Interval Types" on page 121.

Interval names for use with SAS date values can be prefixed with 'DT' to construct interval names for use with SAS datetime values. The interval names DTYEAR, DTSEMIYEAR, DTQTR, DTMONTH, DTSEMIMONTH, DTTENDAY, DTWEEK, DTWEEKDAY, DTDAY, DTYEARV, DTR445YR, DTR454YR, DTR544YR, DTR445QTR, DTR454QTR, DTR544QTR, DTR445MON, DTR454MON, DTR544MON, and DTWEEKV are used with SAS datetime values.

Constructing Interval Names

Multipliers and shift indexes can be used with the basic interval names to construct more complex interval specifications. The general form of an interval name is as follows:

NAMEn.s

The three parts of the interval name are as follows:

NAME	the name of the basic interval type. For example, YEAR specifies yearly intervals.
n	an optional multiplier that specifies that the interval is a multiple of the period of the basic interval type. For example, the interval YEAR2 consists of two-year (biennial) periods.
s	an optional starting subperiod index that specifies that the intervals are shifted to later starting points. For example, YEAR.3 specifies yearly periods shifted to start on the first of March of each calendar year and to end in February of the following year.

Both the multiplier *n* and the shift index *s* are optional and default to 1. For example, YEAR, YEAR1, YEAR.1, and YEAR1.1 are all equivalent ways of specifying ordinary calendar years.

To test for a valid interval specification, use the INTTEST function:

```
interval = 'MONTH3.2';
valid = INTTEST( interval );
valid = INTTEST( 'YEAR4');
```

INTTEST returns a value of 0 if the argument is not a valid interval specification and 1 if the argument is a valid interval specification. The INTTEST function can also be used in a DATA step to test an interval before calling an interval function:

```
valid = INTTEST( interval );
if ( valid = 1 ) then do;
    end_date = INTNX( interval, date, 0, 'E' );
    Status = 'Success';
end;
if ( valid = 0 ) then Status = 'Failure';
```

For more information about the INTTEST function, see the *SAS Functions and CALL Routines: Reference*.

Shifted Intervals

Different kinds of intervals are shifted by different subperiods:

- YEAR, SEMIYEAR, QTR, and MONTH intervals are shifted by calendar months.
- WEEK and DAY intervals are shifted by days.
- SEMIMONTH intervals are shifted by semimonthly periods.
- TENDAY intervals are shifted by 10-day periods.
- YEARV intervals are shifted by WEEKV intervals.
- R445YR, R445QTR, and R445MON intervals are shifted by R445MON intervals.
- R454YR, R454QTR, and R454MON intervals are shifted by R454MON intervals.
- R544YR, R544QTR, and R544MON intervals are shifted by R544MON intervals.
- WEEKV intervals are shifted by days.
- WEEKDAY intervals are shifted by weekdays.
- HOUR intervals are shifted by hours.
- MINUTE intervals are shifted by minutes.
- SECOND intervals are shifted by seconds.

The INTSHIFT function returns the shift interval:

```
interval = 'MONTH3.2';
shift_interval = INTSHIFT( interval );
```

In this example, the value of shift_interval is 'MONTH'. For more information about the INTSHIFT function, see the *SAS Functions and CALL Routines: Reference*.

If a subperiod is specified, the shift index cannot be greater than the number of subperiods in the whole interval. For example, you can use YEAR2.24, but YEAR2.25 is an error because there is no 25th month in a two-year interval.

For interval types that shift by subperiods that are the same as the basic interval type, only multiperiod intervals can be shifted. For example, MONTH type intervals shift by MONTH subintervals; thus, monthly intervals cannot be shifted because there is only one month in MONTH. However, bimonthly intervals can be shifted because there are two MONTH intervals in each MONTH2 interval. The interval name MONTH2.2 specifies bimonthly periods that start on the first day of even-numbered months.

Beginning Dates and Datetimes of Intervals

Intervals that represent divisions of a year begin with the start of the year (1 January). YEARV, R445YR, R454YR, and R544YR intervals begin with the first week of the International Organization for Standardization (ISO) year, the Monday on or immediately preceding January 4th. R445QTR, R454QTR, and R544QTR intervals begin with the 1st, 14th, 27th, and 40th weeks of the ISO year. MONTH2 periods begin with odd-numbered months (January, March, May, and so on).

Likewise, intervals that represent divisions of a day begin with the start of the day (midnight). Thus, HOUR8.7 intervals divide the day into the periods 06:00 to 14:00, 14:00 to 22:00, and 22:00 to 06:00.

Intervals that do not nest within years or days begin relative to the SAS date or datetime value 0. The arbitrary reference time of midnight on January 1, 1960, is used as the origin for nonshifted intervals, and shifted intervals are defined relative to that reference point. For example, MONTH13 defines the intervals that begin January 1, 1960, February 1, 1961, March 1, 1962, and so on, in addition to the intervals that begin December 1, 1958, November 1, 1957, and so on before the base date January 1, 1960.

Similarly, the WEEK2 interval begins relative to the Sunday of the week of January 1, 1960. The interval specification WEEK6.13 defines six-week periods that start on second Fridays, and the convention of counting relative to the period that contains January 1, 1960, indicates the starting date or datetime of the interval closest to January 1, 1960, that corresponds to the second Fridays of six-week intervals.

Intervals always begin on the date or datetime defined by the base interval name, the multiplier, and the shift value. The end of the interval immediately precedes the beginning of the next interval. However, an interval can be identified by any date or datetime value between its starting and ending values, inclusive. For more information about generating identifying dates for intervals, see the section “Alignment of SAS Dates” on page 136.

Summary of Interval Types

The interval types are summarized as follows:

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

YEARV
specifies ISO 8601 yearly intervals. The ISO 8601 year starts on the Monday on or immediately preceding January 4th. 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 further discussion of ISO weeks, see Technical Committee ISO/TC 154 (Processes, Data Elements, and Documents in Commerce, Industry, and Administration) (2004). The starting subperiod *s* is in ISO 8601 weeks (WEEKV).

R445YR
is the same as YEARV except that the starting subperiod *s* is in retail 4-4-5 months (R445MON).

R454YR
is the same as YEARV except that the starting subperiod *s* is in retail 4-5-4 months (R454MON). For a discussion of the retail 4-5-4 calendar, see National Retail Federation (2007).

R544YR
is the same as YEARV except that the starting subperiod *s* 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 *s* 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 *s* 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 *s* 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 a discussion of the retail 4-5-4 calendar, see National Retail Federation (2007). The starting subperiod *s* 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 *s* is in retail 5-4-4 months (R544MON).

MONTH
specifies monthly intervals. Abbreviations are MONTH, MONTHS, MONTHLY, and MON. The starting subperiod *s* 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 *s* 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 a discussion of the retail 4-5-4 calendar, see National Retail Federation (2007). The starting subperiod *s* 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 *s* 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 *s* 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 *s* 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 *s* 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 *s* is in days (DAY). Note that WEEKV differs from WEEK in that WEEKV.1 starts on Monday, WEEKV.2 starts on Tuesday, and so forth.

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 *s* 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 *s* is in days (DAY).

HOUR

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

MINUTE

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

SECOND

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

Examples of Interval Specifications

Table 4.1 shows examples of different kinds of interval specifications.

Table 4.1 Examples of Intervals

Name	Description of Interval
YEAR	Years that start in January
YEAR.10	Years that start in October
YEAR2.7	Biennial intervals that start in July of even years
YEAR2.19	Biennial intervals that start in July of odd years
YEAR4.11	Four-year intervals that start in November of leap years (frequency of U.S. presidential elections)

Table 4.1 *continued*

Name	Description of Interval
YEAR4.35	Four-year intervals that start in November of even years between leap years (frequency of U.S. midterm elections)
YEARV	Years that start on the Monday on or immediately preceding January 4th
YEARV.2	Years that start on the Monday immediately following January 4th
R445MON	Months that start on the 1st, 5th, 9th, 14th, 18th, 22nd, 27th, 31st, 35th, 40th, 44th, and 48th Monday of the year. The 1st Monday is the Monday on or immediately preceding January 4th
R445MON3	Three-month intervals that start on the 1st, 14th, 27th, and 40th Monday of the year. This is equivalent to R445QTR
R445MON3.2	Three-month intervals that start on the 5th, 18th, 31th, and 44th Monday of the year. This is equivalent to R445QTR.2
WEEK	Weekly intervals that start on Sundays
WEEK2	Biweekly intervals that start on first Sundays
WEEK1.1	Same as WEEK
WEEK.2	Weekly intervals that start on Mondays
WEEK6.3	Six-week intervals that start on first Tuesdays
WEEK6.11	Six-week intervals that start on second Wednesdays
WEEKDAY	Daily with Friday-Saturday-Sunday counted as the same day (five-day work week with a Saturday-Sunday weekend)
WEEKDAY17W	Same as WEEKDAY
WEEKDAY5.2	Five weekdays that start on Monday. If WEEKDAY data are accumulated into weekly data, the interval of the accumulated data is WEEKDAY5.2
WEEKDAY67W	Daily with Thursday-Friday-Saturday counted as the same day (five-day work week with a Friday-Saturday weekend)
WEEKDAY1W	Daily with Saturday-Sunday counted as the same day (six-day work week with a Sunday weekend)
WEEKDAY3.2	Three-weekday intervals (with Friday-Saturday-Sunday counted as one weekday) with the cycle three-weekday periods aligned to Monday, January 4, 1960
HOUR8.7	Eight-hour intervals that start at 6 a.m., 2 p.m., and 10 p.m. (might be used for work shifts)

Custom Time Intervals

The standard time intervals described in the previous sections do not always fit the data. For example, you might want to use fiscal months that begin on the 10th of each month, but the MONTH interval begins on the 1st of each month. Or you might collect data hourly for a business that is closed at night, but using the DTHOUR interval results in gaps in the data that can cause problems in standard time series analysis. In another case, you might wish to calculate the number of business days between dates, excluding holidays and weekends, but holidays are counted when you use the INTCK function with the WEEKDAY interval. For more information about the INTCK function, see "Interval Functions INTNX and INTCK" on page 89.

Time series can be analyzed using observation numbers as the identifying reference. However, it is often desirable to maintain the time stamp for other types of modeling such as regression variables based on time or reconciliation.

To address these issues, you can define custom intervals within a given SAS program. The use of custom intervals requires the following two steps for each interval:

1 Associate a data set name with a custom interval name by using the INTERVALDS= system option. For more information about the INTERVALDS= option, see the *SAS System Options: Reference*. The following example associates the data set StoreHoursDS with the custom interval StoreHours:

```
options intervalds=(StoreHours=StoreHoursDS);
```

2 Create a data set that describes the custom interval. The data set must contain a BEGIN variable. It can also contain an END and a SEASON variable. It should contain a FORMAT statement for the BEGIN variable that specifies a SAS date, SAS datetime, or numeric format that matches the BEGIN variable data. If the END variable is present, it should also be included in the FORMAT statement. A numeric format that is not a SAS date or SAS datetime format indicates that the values are observation numbers. If the END variable is not present, then the implied value of END at each observation is one less than the value of BEGIN at the next observation.

   The span of the custom interval data set should include any dates or times that are necessary for performing calculations on the time series, including backcasting, forecasting, and other operations that might extend beyond the series (such as filters).

After the two preceding steps have been completed, the custom interval can be specified in SAS procedures and functions where a standard time interval can be specified.

The following DATA step creates the StoreHoursDS data set, which is appropriate for a business that is open 9 a.m. to 6 p.m. Monday through Friday and 9 a.m. to 1 p.m. Saturday:

```
options intervalds=(StoreHours=StoreHoursDS);
data StoreHoursDS(keep=BEGIN END);
   start = '01JAN2009'D;
   stop  = '31DEC2009'D;
   do date = start to stop;
      dow = WEEKDAY(date);
      datetime=dhms(date,0,0,0);
```

```
      if dow not in (1,7) then
         do hour = 9 to 17;
            begin=intnx('hour',datetime,hour,'b');
            end=intnx('hour',datetime,hour,'e');
            output;
         end;
      else if dow = 7 then
         do hour = 9 to 12;
            begin=intnx('hour',datetime,hour,'b');
            end=intnx('hour',datetime,hour,'e');
            output;
         end;
   end;
   format BEGIN END DATETIME.;
run;

title 'Store Hours Custom Interval';
proc print data=StoreHoursDS(obs=18);
run;
```

The first 18 observations of the custom interval data set are shown in Figure 4.1.

Figure 4.1 Store Hours Custom Interval

Store Hours Custom Interval

Obs	begin	end
1	01JAN09:09:00:00	01JAN09:09:59:59
2	01JAN09:10:00:00	01JAN09:10:59:59
3	01JAN09:11:00:00	01JAN09:11:59:59
4	01JAN09:12:00:00	01JAN09:12:59:59
5	01JAN09:13:00:00	01JAN09:13:59:59
6	01JAN09:14:00:00	01JAN09:14:59:59
7	01JAN09:15:00:00	01JAN09:15:59:59
8	01JAN09:16:00:00	01JAN09:16:59:59
9	01JAN09:17:00:00	01JAN09:17:59:59
10	02JAN09:09:00:00	02JAN09:09:59:59
11	02JAN09:10:00:00	02JAN09:10:59:59
12	02JAN09:11:00:00	02JAN09:11:59:59
13	02JAN09:12:00:00	02JAN09:12:59:59
14	02JAN09:13:00:00	02JAN09:13:59:59
15	02JAN09:14:00:00	02JAN09:14:59:59
16	02JAN09:15:00:00	02JAN09:15:59:59
17	02JAN09:16:00:00	02JAN09:16:59:59
18	02JAN09:17:00:00	02JAN09:17:59:59

The following DATA step creates the FMDS data set to define a custom interval FiscalMonth, which is appropriate for a business that uses fiscal months that start on the 10th of each month. The SAME alignment option of the INTNX function specifies that the dates generated by the INTNX function are the same day of the month as the date in the start variable. For more information about the INTNX function, see "SAS Date, Time, and Datetime Functions" on page 136. The MONTH function assigns the month of the BEGIN variable to the SEASON variable. This specifies monthly seasonality.

```
options intervalds=(FiscalMonth=FMDS);
data FMDS(keep=BEGIN SEASON);
   start = '10JAN1999'D;
   stop  = '10JAN2001'D;
   nmonths = INTCK('MONTH',start,stop);
   do i=0 to nmonths;
      BEGIN = INTNX('MONTH',start,i,'S');
      SEASON = MONTH(BEGIN);
      output;
   end;
   format BEGIN DATE.;
run;
```

The difference between the custom FiscalMonth interval and a standard interval can be seen in the following example. The output shown in Figure 4.2 compares how the data are accumulated. For the FiscalMonth interval, values in the first nine days of the month are accumulated with the interval that begins in the previous month. For the standard MONTH interval, values in the first nine days of the month are accumulated with the calendar month.

```
data sales(keep=DATE sales);
   do date = '01JAN2000'D to '31DEC2000'D;
      month = MONTH(date);
      dayofmonth = DAY(date);
      sales = 0;
      if ( dayofmonth lt 10 ) then sales = month/9;
      output;
   end;
   format date monyy.;
run;

proc timeseries data=sales out=dataInFiscalMonths;
   id DATE interval=FiscalMonth accumulate=total;
   var sales;
run;

proc timeseries data=sales out=dataInStdMonths;
   id DATE interval=Month accumulate=total;
   var sales;
run;

data compare;
     merge dataInFiscalMonths(rename=(sales=FM_sales))
           dataInStdMonths(rename=(sales=SM_sales));
     by DATE;
run;

title 'Standard Monthly Data vs. Fiscal Month Data';
proc print data=compare;
run;
```

Figure 4.2 Fiscal Months Custom Interval

Standard Monthly Data vs. Fiscal Month Data

Obs	date	FM_sales	SM_sales
1	10-DEC-1999	1	.
2	01-JAN-2000	.	1
3	10-JAN-2000	2	.
4	01-FEB-2000	.	2
5	10-FEB-2000	3	.
6	01-MAR-2000	.	3
7	10-MAR-2000	4	.
8	01-APR-2000	.	4
9	10-APR-2000	5	.
10	01-MAY-2000	.	5
11	10-MAY-2000	6	.
12	01-JUN-2000	.	6
13	10-JUN-2000	7	.
14	01-JUL-2000	.	7
15	10-JUL-2000	8	.
16	01-AUG-2000	.	8
17	10-AUG-2000	9	.
18	01-SEP-2000	.	9
19	10-SEP-2000	10	.
20	01-OCT-2000	.	10
21	10-OCT-2000	11	.
22	01-NOV-2000	.	11
23	10-NOV-2000	12	.
24	01-DEC-2000	.	12
25	10-DEC-2000	0	.

The next example uses custom intervals in the time function INTCK to omit holidays when counting business days. The result is shown in Figure 4.3.

```
options intervalds=(BankingDays=BankDayDS);
data BankDayDS(keep=BEGIN);
   start = '15DEC1998'D;
   stop  = '15JAN2002'D;
   nwkdays = INTCK('WEEKDAY',start,stop);
   do i = 0 to nwkdays;
      BEGIN = INTNX('WEEKDAY',start,i);
      year = YEAR(BEGIN);
      if BEGIN ne HOLIDAY("NEWYEAR",year) and
         BEGIN ne HOLIDAY("MLK",year) and
         BEGIN ne HOLIDAY("USPRESIDENTS",year) and
         BEGIN ne HOLIDAY("MEMORIAL",year) and
         BEGIN ne HOLIDAY("USINDEPENDENCE",year) and
         BEGIN ne HOLIDAY("LABOR",year) and
         BEGIN ne HOLIDAY("COLUMBUS",year) and
         BEGIN ne HOLIDAY("VETERANS",year) and
         BEGIN ne HOLIDAY("THANKSGIVING",year) and
```

```
         BEGIN ne HOLIDAY("CHRISTMAS",year) then
      output;
   end;
   format BEGIN DATE.;
run;

data CountDays;
   start = '01JAN1999'D;
   stop  = '31DEC2001'D;
   ActualDays = INTCK('DAYS',start,stop);
   Weekdays   = INTCK('WEEKDAYS',start,stop);
   BankDays   = INTCK('BankingDays',start,stop);
   format start stop DATE.;
run;

title 'Methods of Counting Days';
proc print data=CountDays;
run;
```

Figure 4.3 Bank Days Custom Interval

Methods of Counting Days

Obs	start	stop	ActualDays	Weekdays	BankDays
1	01JAN99	31DEC01	1095	781	757

Date and Datetime Informats

Table 4.2 lists some of the SAS date and datetime informats available to read date, time, and datetime values. For a discussion of the use of date and datetime informats, see Chapter 3, "Working with Time Series Data." For a complete description of these informats, see *SAS Programmers Guide: Essentials*.

For each informat, Table 4.2 shows an example of a date or datetime value written in the style that the informat is designed to read. You can specify the width of each informat by adding *w*. For informats that include second values, you can specify the number of decimal digits for seconds by adding *d*. Table 4.2 shows the width range allowed by the informat and the default width. The date 17 October 1991 and the time 2:25:32 p.m. are used for the example in all cases.

Table 4.2 Frequently Used SAS Date and Datetime Informats

Informat	Example	Description	Width Range	Default Width
ANYDTDTE*w.*		Reads and extracts the date value from any of the following: DATE, DATETIME, DDMMYY, JULIAN, MDYAMPM, MMDDYY, MMxYY*, MONYY, TIME, YMDDTTM, YYMMDD, YYQ, YYxMM*, month-day-year	5–32	9

Table 4.2 *continued*

Informat	Example	Description	Width Range	Default Width
ANYDTDTM*w.*		Reads and extracts the datetime value from any of the following: DATE, DATETIME, DDMMYY, JULIAN, MMDDYY, MMxYY*, MONYY, TIME, YYMMDD, YYQ, YYxMM*, month-day-year	1–32	19
ANYDTTME*w.*		Reads and extracts the time value from any of the following: DATE, DATETIME, DDMMYY, JULIAN, MMDDYY, MONYY, TIME, YYMMDD, YYQ, month-day-year	1–32	8
DATE*w.*	17oct91	Day, month abbreviation, and year: *ddmonyy*	7–32	7
DATETIME*w.d*	17oct91:14:45:32	Date and time: *ddmonyy*:*hh*:*mm*:*ss*	13–40	18
DDMMYY*w.*	17/10/91	Day, month, year: *ddmmyy*, *dd*/*mm*/*yy*, *dd-mm-yy*, or *dd mm yy*	6–32	6
JULIAN*w.*	91290	Year and day of year (Julian dates): *yyddd*	5–32	5
MMDDYY*w.*	10/17/91	Month, day, year: *mmddyy*, *mm*/*dd*/*yy*, *mm-dd-yy*, or *mm dd yy*	6–32	6
MONYY*w.*	Oct91	Month abbreviation and year: *monyy*	5–32	5
NENGO*w.*	H.03/10/17	Japanese Nengo notation	7–32	10
TIME*w.d*	14:45:32	Hours, minutes, seconds: *hh*:*mm*:*ss* or hours, minutes: *hh*:*mm*	5–32	8
WEEKV*w.*	1991-W42-04	ISO 8601 year, week, day of week: *yyyy*-W*ww*-*dd*	3–200	11
YYMMDD*w.*	91/10/17	Year, month, day: *yymmdd*, *yy*/*mm*/*dd*, *yy-mm-dd*, or *yy mm dd*	6–32	6
YYQ*w.*	91Q4	Year and quarter of year: *yy*Q*q*	4–32	4

Date, Time, and Datetime Formats

Some of the commonly used SAS date and datetime formats are listed in Table 4.3 and Table 4.4. You can specify the width value for each format by adding *w*. The tables list the range of width values allowed and the default width value for each format.

The notation used by a format is abbreviated in different ways depending on the width option used. For example, the format MMDDYY8. writes the date 17 October 1991 as 10/17/91, while the format MMDDYY6. writes this date as 101791. In particular, formats that display the year show two-digit or four-digit year values depending on the width option. The examples shown in the tables use the default width.

The interval function INTFMT returns a recommended format for time ID values based on the interval that describes the frequency of the values. The following example uses INTFMT to select a format to display the quarterly time ID variable qtrDate. In this example, INTFMT returns the format YYQC6., which displays the year in four digits and the quarter in a single digit. This selected format is stored in a macro variable that is created by the CALL SYMPUT statement. The second argument to INTFMT controls the width of the year for date formats; it can take the value 'long' or 'l' to indicate 4 for the year width or the value 'short' or 's' to indicate 2 for the year width. For more information about the INTFMT function, see the . For more information about the CALL SYMPUT statement, see the *SAS DATA Step Statements: Reference*.

The macro variable &FMT is then used in the FORMAT statement in the PROC PRINT step as follows:

```
data b(keep=qtrDate);
   interval = 'QTR';
   form = INTFMT( interval, 'long' );
   call symput('fmt',form);
   do i=1 to 4;
      qtrDate = INTNX( interval, '01jan00'd, i-1 );
      output;
   end;
run;

proc print;
   format qtrDate &fmt;
run;
```

It is also possible to display date and datetime values as strings by using the format that is identified by INTFMT. In the following example, INTFIT is used to identify the intervals of the sashelp.citiwk and sashelp.air data sets. Then INTFMT is used to identify formats that are based on the intervals. The formats are then used to convert the first SAS date value of each data set to a string. The START variable displays the date of the first observation of each data set. This method assumes that the interval of the data set can be identified by examining the first two observations. This is often the case for output data sets and data sets that have been properly prepared for input by using a procedure such as the TIMESERIES procedure. More than two observations might be required to identify the difference between a DAY interval and a WEEKDAY interval. This example would need to be modified if the DATE variable contained SAS datetime values.

```
data a(keep=DATE0 DATE INTERVAL FMT START);
   length START INTERVAL FMT $32;
   format date0 date DATE.;
   set sashelp.citiwk(obs=2) sashelp.air(obs=2);
   DATE0 = lag(date);
```

```
   if ( mod(_n_,2) eq 1 ) then delete;
   if ( mod(_n_,2) eq 0 ) then INTERVAL = intfit( DATE0, date, 'D' );
   if ( mod(_n_,2) eq 0 ) then FMT = INTFMT( interval, 'l' );
   START = putn( DATE0, FMT );
run;

proc print;
run;
```

For a complete description of these formats, including the variations of the formats produced by different width options, see *SAS Programmers Guide: Essentials*. For a discussion of the use of date and datetime formats, see Chapter 3, "Working with Time Series Data,".

Date Formats

Table 4.3 lists some of the available SAS date formats. For each format, an example is shown of a date value in the notation produced by the format. The date '17OCT91'D is used as the example.

Table 4.3 Frequently Used SAS Date Formats

Format	Example	Description	Width Range	Default Width
DATE*w.*	17OCT91	Day, month abbreviation, year: *ddmonyy*	5–9	7
DAY*w.*	17	Day of month	2–32	2
DDMMYY*w.*	17/10/91	Day, month, year: *dd*/*mm*/*yy*	2–8	8
DOWNAME*w.*	Thursday	Name of day of the week	1–32	9
JULDAY*w.*	290	Day of year	3–32	3
JULIAN*w.*	91290	Year and day of year: *yyddd*	5–7	5
MMDDYY*w.*	10/17/91	Month, day, year: *mm*/*dd*/*yy*	2–8	8
MMYY*w.*	10M1991	Month and year: *mm*M*yyyy*	5–32	7
MMYYC*w.*	10:1991	Month and year: *mm*:*yyyy*	5–32	7
MMYYD*w.*	10-1991	Month and year: *mm*-*yyyy*	5–32	7
MMYYP*w.*	10.1991	Month and year: *mm*.*yyyy*	5–32	7
MMYYS*w.*	10/1991	Month and year: *mm*/*yyyy*	5–32	7

Table 4.3 *continued*

Format	Example	Description	Width Range	Default Width
MMYYN*w.*	101991	Month and year: *mmyyyy*	5–32	6
MONNAME*w.*	October	Name of month	1–32	9
MONTH*w.*	10	Month of year	1–32	2
MONYY*w.*	OCT91	Month abbreviation and year: *monyy*	5–7	5
QTR*w.*	4	Quarter of year	1–32	1
QTRR*w.*	IV	Quarter in roman numerals	3–32	3
NENGO*w.*	H.03/10/17	Japanese Nengo notation	2–10	10
WEEKDATE*w.*	Thursday, October 17, 1991	*day-of-week*, *month-name dd*, *yyyy*	3–37	29
WEEKDATX*w.*	Thursday, 17 October 1991	*day-of-week*, *dd month-name yyyy*	3–37	29
WEEKDAY*w.*	5	Day of week	1–32	1
WEEKV*w.*	1991-W42-04	ISO 8601 year, week, day of week: *yyyy*-W*ww*-*dd*	3–200	11
WORDDATE*w.*	October 17, 1991	*month-name dd*, *yyyy*	3–32	18
WORDDATX*w.*	17 October 1991	*dd month-name yyyy*	3–32	18
YEAR*w.*	1991	Year: *yyyy*	2–32	4
YYMM*w.*	1991M10	Year and month: *yyyy*M*mm*	5–32	7
YYMMC*w.*	1991:10	Year and month: *yyyy*:*mm*	5–32	7
YYMMD*w.*	1991-10	Year and month: *yyyy-mm*	5–32	7
YYMMP*w.*	1991.10	Year and month: *yyyy.mm*	5–32	7
YYMMS*w.*	1991/10	Year and month: *yyyy*/*mm*	5–32	7
YYMMN*w.*	199110	Year and month: *yyyymm*	5–32	7

Table 4.3 *continued*

Format	Example	Description	Width Range	Default Width
YYMON*w.*	1991OCT	Year and month abbreviation: *yyyymon*	5–32	7
YYMMDD*w.*	91/10/17	Year, month, day: *yy*/*mm*/*dd*	2–8	8
YYQ*w.*	1991Q4	Year and quarter: *yyyy*Q*q*	4–6	6
YYQC*w.*	1991:4	Year and quarter: *yyyy*:*q*	4–32	6
YYQD*w.*	1991-4	Year and quarter: *yyyy*-*q*	4–32	6
YYQP*w.*	1991.4	Year and quarter: *yyyy*.*q*	4–32	6
YYQS*w.*	1991/4	Year and quarter: *yyyy*/*q*	4–32	6
YYQN*w.*	19914	Year and quarter: *yyyyq*	3–32	5
YYQR*w.*	1991QIV	Year and quarter in roman numerals: *yyyy*Q*rr*	6–32	8
YYQRC*w.*	1991:IV	Year and quarter in roman numerals: *yyyy*:*rr*	6–32	8
YYQRD*w.*	1991-IV	Year and quarter in roman numerals: *yyyy*-*rr*	6–32	8
YYQRP*w.*	1991.IV	Year and quarter in roman numerals: *yyyy*.*rr*	6–32	8
YYQRS*w.*	1991/IV	Year and quarter in roman numerals: *yyyy*/*rr*	6–32	8
YYQRN*w.*	1991IV	Year and quarter in roman numerals: *yyyyrr*	6–32	8

Datetime and Time Formats

Table 4.4 lists some of the available SAS datetime and time formats. For each format, the example shows the formatted value. The value of the variable dt is '17OCT91:14:25:32'DT. You can specify the width of each format by adding *w*. For formats that allow a decimal value, you can specify the number of decimal digits by adding *d*.

Table 4.4 Frequently Used SAS Datetime and Time Formats

Format	Value	Example	Description	Width Range	Default Width
DATETIME*w.d*	dt	17OCT91:14:25:32	*ddmonyy*: *hh*:*mm*:*ss.ss*	7–40	16
DTWKDATX*w*.	dt	Thursday, 17 October 1991	*day-of-week*, *dd month yyyy*	3–37	29
HHMM*w.d*	TIMEPART(dt)	14:26	Hour and minute: *hh*:*mm.mm*	2–20	5
HOUR*w.d*	TIMEPART(dt)	14	Hour: *hh.hh*	2–20	2
MMSS*w.d*	HMS(0, MINUTE(dt), SECOND(dt))	25:32	Minutes and seconds: *mm*:*ss.ss*	2–20	5
TIME*w.d*	TIMEPART(dt)	14:25:32	Time of day: *hh*:*mm*:*ss.ss*	2–20	8
TOD*w.d*	dt	14:25:32	Time of day: *hh*:*mm*:*ss.ss*	2–20	8

Alignment of SAS Dates

SAS date values that are used to identify time series observations produced by SAS/ETS and SAS High-Performance Forecasting procedures are normally aligned with the beginning of the time intervals that correspond to the observations. For example, for monthly data for 1994, the date values that identify the observations are 1Jan94, 1Feb94, 1Mar94, . . . , 1Dec94.

However, for some applications it might be preferable to use end-of-period dates, such as 31Jan94, 28Feb94, 31Mar94, . . . , 31Dec94. For other applications, such as plotting time series, it might be more convenient to use interval midpoint dates to identify the observations.

Many SAS/ETS and SAS High-Performance Forecasting procedures provide an ALIGN= option to control the alignment of dates for outputting time series observations. SAS/ETS procedures that support the ALIGN= option are ARIMA, DATASOURCE, ESM, EXPAND, SIMILARITY, TIMESERIES, UCM, and VARMAX. SAS High-Performance Forecasting procedures that support the ALIGN= option are HPFRECONCILE, HPF, HPFDIAGNOSE, HPFENGINE, and HPFEVENTS.

ALIGN=

The ALIGN= option can have the following values:

BEGINNING	specifies that dates be aligned to the start of the interval. This is the default. BEGINNING can be abbreviated as BEGIN, BEG, or B.
MIDDLE	specifies that dates be aligned to the interval midpoint, the average of the beginning and ending values. MIDDLE can be abbreviated as MID or M.
ENDING	specifies that dates be aligned to the end of the interval. ENDING can be abbreviated as END or E.

For information about the calculation of the beginning and ending values of intervals, see the section "Beginning Dates and Datetimes of Intervals" on page 120.

SAS Date, Time, and Datetime Functions

SAS date, time, and datetime functions are used to perform the following tasks:

- compute date, time, and datetime values from calendar and time-of-day values
- compute calendar and time-of-day values from date and datetime values
- convert between date, time, and datetime values
- perform calculations that involve time intervals
- provide information about time intervals
- provide information about seasonality

For all interval functions, you can supply the intervals and other character arguments either directly as a quoted string or as a SAS character variable. When you use a character variable, you should set the length of the character variable to at least the length of the longest string for that variable that is used in the DATA step.

Also, to ensure correct results when using interval functions, use date intervals with date values and datetime intervals with datetime values.

For a complete description of these functions, see *SAS Functions and CALL Routines: Reference*.

The following list shows SAS date, time, and datetime functions in alphabetical order:

DATE()
returns today's date as a SAS date value.

DATEJUL(*yyddd* **)**
returns the SAS date value when given the Julian date in *yyddd* or *yyyyddd* format. For example, `DATE = DATEJUL(99001);` assigns the SAS date value '01JAN99'D to DATE, and `DATE = DATEJUL(1999365);` assigns the SAS date value '31DEC1999'D to DATE.

DATEPART(*datetime* **)**
returns the date part of a SAS datetime value as a date value.

DATETIME()
returns the current date and time of day as a SAS datetime value.

DAY(*date* **)**
returns the day of the month from a SAS date value.

DHMS(*date, hour, minute, second* **)**
returns a SAS datetime value for date, hour, minute, and second values.

FMTINFO('*format-name***', '***information-type***')**
returns the information specified by *information-type* for *format-name*. This function is useful for determining the category of a variable if the variable has a format. Specifying the *information-type* as 'cat' returns the category of the format. Examples of categories are date, datetime, time, and num. For example, `FMTINFO('MMDDYY','cat')` returns 'date'.

HMS(*hour, minute, second* **)**
returns a SAS time value for hour, minute, and second values.

HOLIDAY('*holiday***',** *year* **)**
returns a SAS date value for the holiday and year specified. Valid values for holiday are 'BOXING', 'CANADA', 'CANADAOBSERVED', 'CHRISTMAS', 'COLUMBUS', 'EASTER', 'FATHERS', 'HALLOWEEN', 'JUNETEENTH', 'JUNETEENTHUSG', 'JUNETEENTHUSPS', 'LABOR', 'MLK', 'MEMORIAL', 'MOTHERS', 'NEWYEAR','THANKSGIVING', 'THANKSGIVINGCANADA', 'USINDEPENDENCE', 'USPRESIDENTS', 'VALENTINES', 'VETERANS', 'VETERANSUSG', 'VETERANSUSPS', and 'VICTORIA'. For example: `EASTER2000 = HOLIDAY('EASTER', 2000);`

HOUR(*datetime* **)**
returns the hour from a SAS datetime or time value.

INTCINDEX('*date-interval***',** *date* **)**

INTCINDEX('*datetime-interval***',** *datetime* **)**
returns the index of the seasonal cycle when given an interval and an appropriate SAS date, datetime, or time value. For example, the seasonal cycle for INTERVAL='DAY' is 'WEEK', so `INTCINDEX('DAY','01SEP78'D);` returns 35 because September 1, 1978, is the sixth day of the 35th week of the year. For correct results, date intervals should be used with date values, and datetime intervals should be used with datetime values.

INTCK('*date-interval***',** *date1, date2* **<,** *'method'* **>)**

INTCK('*datetime-interval***',** *datetime1, datetime2* **<,** *'method'* **>)**
returns the number of boundaries of intervals of the given kind that lie between the two date or datetime values. The optional method argument specifies that the intervals are counted using either a discrete or a continuous method. The default DISCRETE (or DISC or D) method uses discrete time intervals. For the DISCRETE method, the distance in MONTHS between January 31, 2000, and February 1, 2000, is one month. The CONTINUOUS (or CONT or C) method uses continuous time intervals. For the CONTINUOUS method, the distance in MONTHS between January 15, 2000, and February 14, 2000, is zero, but the distance in MONTHS between January 15, 2000, and February 15, 2000, is one month.

INTCYCLE('*interval***' <,** *seasonality* **>)**
returns the interval of the seasonal cycle, given a date, time, or datetime interval. For example, INTCYCLE('MONTH') returns 'YEAR' because the months January, February, . . . , December constitute a yearly cycle. INTCYCLE('DAY') returns 'WEEK' because Sunday, Monday, . . . , Saturday constitute a weekly cycle.

You can specify the optional *seasonality* argument to construct a cycle other than the default seasonal cycle. For example, INTCYCLE('MONTH', 3) returns 'QTR'. The optional second argument is the seasonal frequency.

INTFIT(*date1, date2***, '***D***')**

INTFIT(*datetime1, datetime2***, '***DT***')**

INTFIT(*obs1, obs2***, '***OBS***')**
returns an interval that fits exactly between two SAS date, datetime, or observation values. That is, if the interval result of the INTFIT function is used with *date1*, 1, and SAMEDAY alignment in the INTNX function, then the result is *date2*. This concept is illustrated in the following example, where result1 is the same as date1 and result2 is the same as date2:

```
FitInterval = INTFIT( date1, date2, 'D' );
result1 = INTNX( FitInterval, date1, 0, 'SAMEDAY');
result2 = INTNX( FitInterval, date1, 1, 'SAMEDAY');
```

More than one interval can fit the preceding definition. For example, two SAS date values that are seven days apart could be fit with either 'DAY7' or 'WEEK'. The INTFIT function chooses the more common interval, so 'WEEK' is the result when the dates are seven days apart. The INTFIT function can be used to detect the possible frequency of the time series or to analyze frequencies of other events in a time series, such as outliers or missing values.

INTFMT('*interval***' ,'***size***')**

returns a recommended format when given a date, time, or datetime interval for displaying the time ID values associated with a time series of the given interval. The second argument to INTFMT controls the width of the year for date formats; it can take the value 'long' or 'l' to specify that the returned format display a four-digit year or the value 'short' or 's' to specify that the returned format display a two-digit year.

INTGET(*date1, date2, date3* **)**

INTGET(*datetime1, datetime2, datetime3* **)**

returns an interval that fits three consecutive SAS date or datetime values. The INTGET function examines two intervals: the first interval between *date1* and *date2*, and the second interval between *date2* and *date3*. In order for an interval to be detected, either the two intervals must be the same or one interval must be an integer multiple of the other interval. That is, INTGET assumes that at least two of the dates are consecutive points in the time series, and that the other two dates are also consecutive or represent the points before and after missing observations. The INTGET function assumes that large values are SAS datetime values, which are measured in seconds, and that smaller values are SAS date values, which are measured in days. The INTGET function can be used to detect the possible frequency of the time series or to analyze frequencies of other events in a time series, such as outliers or missing values.

INTINDEX('*date-interval***',** *date* **<,** *seasonality* **>)**

INTINDEX('*datetime-interval***',** *datetime* **<,** *seasonality* **>)**

returns the seasonal index for the specified date or datetime interval and an appropriate date or datetime value. The seasonal index is a number that represents the position of the date or datetime value in the seasonal cycle of the specified interval. For example, `INTINDEX('MONTH','01DEC2000'D);` returns 12 because monthly data is yearly periodic and DECEMBER is the 12th month of the year. However, `INTINDEX('DAY','01DEC2000'D);` returns 6 because daily data is weekly periodic and December 01, 2000, is a Friday, the sixth day of the week. To correctly identify the seasonal index, the interval specification should agree with the date or datetime value. For example, `INTINDEX('DTMONTH','01DEC2000'D);` and `INTINDEX('MONTH','01DEC2000:00:00:00'DT);` do not return the expected value of 12. However, both `INTINDEX('MONTH','01DEC2000'D);` and `INTINDEX('DTMONTH','01DEC2000:00:00:00'DT);` return the expected value of 12.

You can specify the optional *seasonality* argument to use a seasonal cycle other than the default seasonal cycle. For example, `INTINDEX('MONTH','01APR2000'D);` returns the value 4, to indicate the fourth month of the year. However, `INTINDEX('MONTH','01APR2000'D,3);` and `INTINDEX('MONTH','01APR2000'D,'QTR');` return the value 1 to indicate the first month of the quarter. Specifying either 3 or 'QTR' for the third argument uses a quarterly seasonal cycle instead of the default yearly seasonal cycle.

INTNEST('*interval***', '***interval***')**

An interval is said to *nest* within another interval if a whole number of the first interval spans the same time period as the second interval for all time periods. For example, DAY is nested within WEEK because there are exactly seven DAY periods within each WEEK for every span of time. However, WEEK is not nested in MONTH because a MONTH period is not consistently a multiple of 7 days. In order to nest, the two intervals must also generate beginning and ending dates that align. For example, WEEK.2 will not nest within WEEK because WEEK begins on Sundays and WEEK.2 begins on Mondays.

The INTNEST function calculates the number of whole periods of the smaller interval that will fit into the period of the larger interval. If the first interval specified spans a larger time period than the second interval specified, then the number returned is positive. If the second interval specified spans a larger period than the first interval specified, then the number returned is negative. For example, `INTNEST('WEEK','DAY')` returns 7, and `INTNEST('DAY','WEEK')` returns –7. A missing value is returned if neither interval nests into the other. Table 4.5 lists the types of results returned by INTNEST and describes how to interpret each result.

Table 4.5 Results Returned by the INTNEST Function

Result	Description	Explanation Example
0	Same	The two input intervals define the same time periods for all time periods. INTNEST('MONTH12','YEAR')
1	Variable number	The first interval contains a whole number of periods of the second interval, but the number varies over time. INTNEST('MONTH','DAY')
-1	Variable number	The second interval contains a whole number of periods of periods of the first interval, but the number varies over time. INTNEST('DAY','YEAR')
$n > 1$	Fixed number	The first interval contains a whole number n periods of the second interval, and that is fixed for all time. INTNEST('WEEK', 'DAY')
$n < -1$	Fixed number	The second interval contains a whole number $-n$ periods of the first interval, and that is fixed for all time. INTNEST('DTHOUR', 'DAY')
Missing value of M	Multiple mismatch	Neither interval will nest into other interval. However, intervals of these types can nest for some multiple values. INTNEST('SEMIMONTH3', 'MONTH')
Missing value of S	Shift mismatch	Neither interval will nest into other interval. However, if a shift value were changed, then the intervals would be the same or one would nest into the other. INTNEST('SEMIMONTH2.2', 'MONTH')
Missing value of B	Base mismatch	The interval bases define time periods that are so different that nesting is not possible for any multiple or shift. For example, YEAR always begins on January 1st of each year, and is shifted by months. However, YEARV always begins on the Monday on or immediately preceding January 4th, and YEARV is shifted by ISO 8601 weeks that begin on Monday. Since January 1st is only a Monday for some years, the intervals will not consistently start on the same day. The same problem exists if the YEAR interval is shifted by months, since the first of a month would not be a Monday for all years. INTNEST('YEAR', 'YEARV')

The result returned by the INTNEST function is of interest when performing the following tasks related to time series:

accumulation	when one interval nests into another interval, even with a variable number, accumulation from the smaller time periods into the larger time periods can be accomplished with a simple rule. If the intervals do not nest, you should consider transforming a time series from one frequency to another with a more complex rule, for example an interpolation.
seasonality	many seasonal models require the higher frequency interval nest into the lower frequency seasonal interval with a fixed number of periods.
time reconciliation	time reconciliation requires that the higher frequency interval nest into the lower frequency interval.

The following example illustrates the relationship between two intervals that nest and both intervals are either date intervals or both intervals are datetime intervals. In this example, for each observation, the value calculated for begin1 is the same as begin2 and the value calculated for end1 is the same as end2:

```
/* interval1 and interval2 are any 2 valid intervals */
nest=INTNEST(interval1,interval2);
/* If interval1 and interval2 are date intervals, then start and end are any
   SAS date values. If interval1 and interval2 are datetime intervals,
   then start and end are SAS datetime values.
   This algorithm would need to be modified if a SAS date interval is
   compared to a SAS datetime interval. */
do date=start to end;
   if ( ( nest = .B ) or
        ( nest = .M ) or
        ( nest = .S )    ) then do;
      /* skip this case as the rule does not apply */
      end;
   else if ( nest = 0 ) then do;
      begin1=INTNX(interval1,date,0);
      begin2=INTNX(interval2,date,0);
      end1=INTNX(interval1,date,nest,'E');
      end2=INTNX(interval2,date,nest,'E');
      end;
   else if ( nest = 1 ) then do;
      begin1=INTNX(interval1,date,0);
      end1=INTNX(interval1,date,0,'E');
      n=INTCK(interval2,begin1,end1);
      begin2=INTNX(interval2,begin1,0);
      end2=INTNX(interval2,begin2,n,'E');
      end;
   else if ( nest = -1 ) then do;
      begin2=INTNX(interval2,date,0);
      end2=INTNX(interval2,date,0,'E');
      n=INTCK(interval1,begin2,end2);
      begin1=INTNX(interval1,begin2,0);
```

```
      end1=INTNX(interval1,begin1,n,'E');
      end;
   else if ( nest > 1 ) then do;
      begin1=INTNX(interval1,date,0);
      begin2=INTNX(interval2,begin1,0);
      end1=INTNX(interval1,date,0,'E');
      end2=INTNX(interval2,begin2,nest-1,'E');
      end;
   else if ( nest < 1 ) then do;
      begin2=INTNX(interval2,date,0);
      begin1=INTNX(interval1,begin2,0);
      end1=INTNX(interval1,begin1,(-nest)-1,'E');
      end2=INTNX(interval2,date,0,'E');
      end;
   output;
end;
```

INTNX('*date-interval***', *date*, *n* < , '*alignment*' >)**

INTNX('*datetime-interval***', *datetime*, *n* < , '*alignment*' >)**

returns the date or datetime value of the beginning of the interval that is *n* intervals from the interval that contains the given date or datetime value. The optional *alignment* argument specifies that the returned date is aligned to the beginning, middle, or end of the interval. Beginning is the default. In addition, you can specify SAME (S) alignment. The SAME alignment bases the alignment of the calculated date or datetime value on the alignment of the input date or datetime value. As illustrated in the following example, the SAME alignment can be used to calculate the meaning of "same day next year" or "same day two weeks from now":

```
nextYear = INTNX( 'YEAR', '15Apr2007'D, 1, 'S' );
TwoWeeks = INTNX( 'WEEK', '15Apr2007'D, 2, 'S' );
```

The preceding example returns '15Apr2008'D for nextYear and '29Apr2007'D for TwoWeeks.

For all values of alignment, the number of discrete intervals *n* between the input date and the resulting date agrees with the input value. In the following example, the result is always that n2 = n1:

```
date2 = INTNX( interval, date1, n1, align );
n2 = INTCK( interval, date1, date2 );
```

The preceding example uses the DISCRETE method of the INTCK function by default. The result n2 = n1 does not always apply when the CONTINUOUS method of the INTCK function is specified.

INTSEAS('*interval***' < , *seasonality* >)**

returns the length of the seasonal cycle for the specified date or datetime interval. The length of a seasonal cycle is the number of intervals in a seasonal cycle. For example, when the interval for a time series is described as monthly, many procedures use the option INTERVAL=MONTH to indicate that each observation in the data corresponds to a particular month. Monthly data are considered to be periodic for a one-year seasonal cycle. There are 12 months in one year, so the number of intervals (months) in a seasonal cycle (year) is 12. For quarterly data, there are 4 quarters in one year, so the number of intervals in a seasonal cycle is 4. The periodicity is not always one year. For example,

INTERVAL=DAY is considered to have a seasonal cycle of one week, and because there are 7 days in a week, the number of intervals in a seasonal cycle is 7.

You can specify the optional *seasonality* argument to use a seasonal cycle other than the default seasonal cycle. For example, INTSEAS('MONTH', 3) and INTSEAS('MONTH', 'QTR') both specify a quarterly seasonal cycle and return the value 3. If the optional *seasonality* argument is numeric, it is the seasonal frequency. If the optional *seasonality* argument is character, it is the seasonal cycle.

INTSHIFT('*interval***')**
returns the shift interval that applies to the shift index if a subperiod is specified. For example, YEAR intervals are shifted by MONTH, so INTSHIFT('YEAR') returns 'MONTH'.

INTTEST('*interval***')**
returns 1 if the interval name is valid, 0 otherwise. For example, `VALID = INTTEST('MONTH');` should set VALID to 1, while `VALID = INTTEST('NOTANINTERVAL');` should set VALID to 0. The INTTEST function can be useful in verifying which values of multiplier *n* and the shift index *s* are valid in constructing an interval name.

JULDATE(*date* **)**
returns the Julian date from a SAS date value. The format of the Julian date is either *yyddd* or *yyyyddd* depending on the value of the system option YEARCUTOFF=. For example, using the default system option values, `JULDATE('31DEC1999'D );` returns 99365, while `JULDATE('31DEC1899'D);` returns 1899365.

MDY(*month, day, year* **)**
returns a SAS date value for month, day, and year values.

MINUTE(*datetime* **)**
returns the minute from a SAS time or datetime value.

MONTH(*date* **)**
returns the numerical value for the month of the year from a SAS date value. For example, `MONTH=MONTH('01JAN2000'D);` returns 1, the numerical value for January.

NWKDOM(*n, weekday, month, year* **)**
returns a SAS date value for the *n*th weekday of the month and year specified. For example, Thanksgiving is always the fourth (*n*=4) Thursday (*weekday*=5) in November (*month*=11). Thus `THANKS2000 = NWKDOM( 4, 5, 11, 2000);` returns the SAS date value for Thanksgiving in the year 2000. The last weekday of a month can be specified by using *n*=5. Memorial Day in the United States is the last (*n*=5) Monday (*weekday*=2) in May (*month*=5), and so `MEMORIAL2002 = NWKDOM( 5, 2, 5, 2002);` returns the SAS date value for Memorial Day in 2002. Because *n*=5 always specifies the last occurrence of the month and most months have only 4 instances of each day, the result for *n*=5 is often the same as the result for *n*=4. NWKDOM is useful for calculating the SAS date values of holidays that are defined in this manner.

QTR(*date* **)**
returns the quarter of the year from a SAS date value.

SECOND(*date* **)**
returns the second from a SAS time or datetime value.

TIME()
returns the current time of day.

TIMEPART(*datetime* **)**
returns the time part of a SAS datetime value.

TODAY()
returns the current date as a SAS date value. (TODAY is another name for the DATE function.)

WEEK(*date* < , '*descriptor*' > **)**
returns the week of year from a SAS date value. The algorithm used to calculate the week depends on the *descriptor*, which can take the value 'U', 'V', or 'W'.

If the descriptor is 'U,' weeks start on Sunday and the range is 0 to 53. If weeks 0 and 53 exist, they are only partial weeks. Week 52 can be a partial week.

If the descriptor is 'V', the result is equivalent to the ISO 8601 week of year definition. The range is 1 to 53. Week 53 is a leap week. The first week of the year, Week 1, and the last week of the year, Week 52 or 53, can include days in another Gregorian calendar year.

If the descriptor is 'W', weeks start on Monday and the range is 0 to 53. If weeks 0 and 53 exist, they are only partial weeks. Week 52 can be a partial week.

WEEKDAY(*date* **)**
returns the day of the week from a SAS date value. The WEEKDAY function produces an integer that represents the day of the week, where 1=Sunday, 2=Monday, ..., 7=Saturday. For example `WEEKDAY=WEEKDAY('17OCT1991'D);` returns 5, the numerical value for Thursday.

YEAR(*date* **)**
returns the year from a SAS date value.

YYQ(*year, quarter* **)**
returns a SAS date value for year and quarter values.

References

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.