SAS® Functions *by* Example

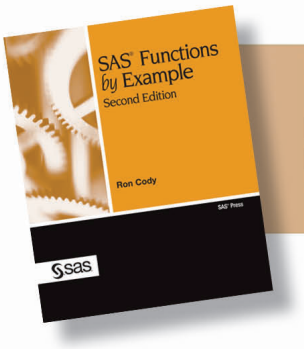
Second Edition

Ron Cody

SAS® Press

sas

Contents



Chapter 4

Date and Time Functions

Introduction

Before you start working with SAS date and time functions, remember that **SAS date values** are the number of days between January 1, 1960, and a specified date. Dates after January 1, 1960, are stored as positive numbers; dates before January 1, 1960, are stored as negative numbers. **SAS time values** are the number of seconds between midnight of the current day and another time value. **SAS datetime values** are the number of seconds between midnight, January 1, 1960, and the specified date and time. Some of the more commonly used date functions extract the day of the week, the month, or the year from a SAS date value.

Other functions deal with intervals, either the number of intervals between two dates or the date after a given number of intervals have passed. You can even compute the number of working days (the default is Saturday and Sunday as non-working days) between two dates. Making this calculation even more useful is the HOLIDAY function that, given a year, returns the date for many of the major holidays.

For situations where you only have month, day, and year values but do not have a SAS date, the MDY function can create a SAS date value, given a value for the month, day, and year. Now let's get started.

Functions That Create SAS Date, Datetime, and Time Values

The first three functions in this group of functions create SAS date values, datetime values, and time values from the constituent parts (month, day, year, hour, minute, second). The DATE and TODAY functions are equivalent and they both return the current date. The DATETIME and TIME functions are used to create SAS datetime and time values, respectively.

Function: MDY

Purpose: To create a SAS date from the month, day, and year.

Syntax: **`MDY(`*`month, day, year`*`)`**

`month` is a numeric variable or constant representing the month of the year (a number from 1 to 12).

day is a numeric variable or constant representing the day of the month (a number from 1 to 31).

year is a numeric variable or constant representing the year.

Values of month, day, and time that do not define a valid date result in a missing value, and an error message is written to the SAS log.

Examples

For these examples, `M = 11, D = 15, Y = 2003.`

Function	Returns
`MDY(M,D,Y)`	`16024 (15NOV2003` – formatted value)
`MDY(10,21,1980)`	`7599 (21OCT1980` – formatted value)
`MDY(1,1,1950)`	`-3652 (01JAN1950` – formatted value)
`MDY(13,01,2003)`	numeric missing value

Program 4.1: Creating a SAS date value from separate variables representing the day, month, and year of the date

```
***Primary function: MDY;

data funnydate;
   input @1  Month  2.
         @7  Year   4.
         @13 Day    2.;
   Date = mdy(Month,Day,Year);
   format Date mmddyy10.;
datalines;
05    2000  25
11    2001  02
;
title "Listing of FUNNYDATE";
proc print data=funnydate noobs;
run;
```

Explanation

Here the values for month, day, and year were not in a form where any of the standard date informats could be used. Therefore, the day, month, and year values were read into separate variables and the MDY function was used to create a SAS date. See the following listing:

```
Listing of FUNNYDATE

Month    Year    Day          Date

   5     2000     25    05/25/2000
  11     2001      2    11/02/2001
```

Program 4.2: Program to read in dates and set the day of the month to 15 if the day is missing from the date

```
***Primary function: MDY;
***Other functions:  SCAN, INPUT, MISSING;

data missing;
   input @1 Dummy $10.;
   Day = scan(Dummy,2,'/');
   if not missing(Day)then Date = input(Dummy,mmddyy10.);
   else Date = mdy(input(scan(Dummy,1,'/'),2.),
                   15,
              input(scan(Dummy,3,'/'),4.));
   format date date9.;
datalines;
10/21/1946
1/  /2000
01/  /2002
;
title "Listing of MISSING";
proc print data=missing noobs;
run;
```

Explanation

This program reads in a date and, when the day of the month is missing, it uses the 15th of the month. If the date was already stored as a character string in a SAS data set, this approach would work well.

The entire date is first read as a character string as the variable DUMMY. Next, the SCAN function is executed with the slash character (/) as the "word" delimiter. The second word is the month. If this is not missing, the INPUT function is used to convert the character string into a SAS date.

If DAY is missing, the MDY function is used to create the SAS date, with the value of 15 representing the day of the month. The listing follows:

```
Listing of MISSING

  Dummy       Day         Date

10/21/1946    21      21OCT1946
1/   /2000              15JAN2000
01/   /2002             15JAN2002
```

Function: DHMS

Purpose: To create a SAS datetime value from a SAS date value and a value for the hour, minute, and second.

Syntax: **DHMS(*date, hour, minute, second*)**

date is a SAS date value, either a variable or a date constant.

hour is a numerical value for the hour of the day. If hour is greater than 24, the function will return the appropriate datetime value.

minute is a numerical value for the number of minutes.

second is a numerical value for the number of seconds.

Values of the date value that are invalid result in a missing value, and an error message is written to the SAS log.

Examples

For these examples, `DATE = '02JAN1960'D, H = 23, M = 15, S = 30.`

Function	Returns
`DHMS(DATE,H,M,S)`	`170130 (02JAN60:23:15:30 –` formatted)
`DHMS('04JUN2003'd,25,12,12)`	`1370394732 (05JUN03:01:12:12 –` formatted)
`DHMS('01JAN1960'd,0,70,0)`	`4200 (01JAN60:01:10:00 –` formatted)

See Program 4.3.

Function: HMS

Purpose: To create a SAS time value from the hour, minute, and second.

Syntax: **HMS(*hour*, *minute*, *second*)**

hour is the value corresponding to the number of hours.

minute is the value corresponding to the number of minutes.

second is the value corresponding to the number of seconds.

Examples

For these examples, `H = 1, M = 30, S = 15.`

Function	Returns
`HMS(H, M, S)`	`5415 (1:30:15 –` formatted value)
`HMS(0, 0, 23)`	`23 (0:00:23 –` formatted value)

See Program 4.3.

Function: DATE and TODAY (equivalent functions)

Purpose: To return the current date.

Syntax: **DATE()** or **TODAY()**

Note that the parentheses are needed even though these functions do not take any arguments. (What did the TODAY function say to the MEAN function? "Don't give me any arguments!")

Examples

Note: This function was run on June 4, 2003.

Function	Returns
`DATE()`	`15860` (`04JUN2003` – formatted)
`TODAY()`	`15860` (`04JUN2003` – formatted)

See Program 4.3.

Function: DATETIME

Purpose: To return the datetime value for the current date and time.

Syntax: **`DATETIME()`**

Examples

Note: This function was run at 8:10 PM on June 4, 2004.

Function	Returns
`DATETIME()`	`1370376600 (04JUN03:20:10:00` – formatted)

See Program 4.3.

Function: TIME

Purpose: To return the time of day when the program was run.

Syntax: **`TIME()`**

Examples

Note: This function was run at 8:10 PM.

Function	Returns
`TIME()`	`72600 (20:10:00` – formatted)

Program 4.3: Determining the date, datetime value, and time of day

```
***Primary functions: DHMS, HMS, TODAY, DATETIME, TIME, YRDIF
***Other functions: INT;

data test;
   Date = today();
   DT = datetime();
   Time = time();
   DT2 = dhms(Date,8,15,30);
   Time2 = hms(8,15,30);
   DOB = '01jan1960'd;
   Age = int(yrdif(DOB,Date,'actual'));
   format Date DOB date9. DT DT2 datetime. Time Time2 time.;
run;

title "Listing of Data Set TEST";
proc print data=test noobs;
run;
```

Explanation

This program was run in the morning of November 10, 2009, so the values for the date, datetime, and time values correspond to that date and time.

The variable DT2 is a SAS datetime value created from the current date and specified values for the hour, minute, and second. TIME2 is a SAS time value created from three values for hour, minute, and second.

Finally, the age was computed using the YRDIF function. (See details and an important note on the YRDIF function later in this chapter.) The INT function was used to compute age as of the last birthday (it throws away all digits to the right of the decimal point). Please see the following listing:

```
Listing of Data Set TEST

     Date            DT             Time            DT2

10JUL2009    10JUL09:09:09:06    9:09:06    10JUL09:08:15:30

 Time2           DOB     Age

8:15:30    01JAN1960      49
```

Creating a Data Set to Demonstrate Other Date Functions

Run Program 4.4 to create a SAS data set called DATES. A listing of this data set follows.

Program 4.4: Program to create the DATES data set

```
data dates;
   informat Date1 Date2 date9.;
   input Date1 Date2;
   format Date1 Date2 date9.;
datalines;
01JAN1960 15JAN1960
02MAR1961 18FEB1962
25DEC2000 03JAN2001
01FEB2002 31MAR2002
;
title "Listing of Data Set DATES";
proc print data=dates noobs;
run;
```

Explanation

Although this is not a function example program, one feature should be explained: Since the INPUT statement is reading list input (i.e., one or more spaces between the data values) and since you need to supply an informat so that the values will be read as SAS date values, an INFORMAT statement precedes the INPUT statement, indicating that both variables, DATE1 and DATE2, should be read with the DATE9. informat.

```
Listing of Data Set DATES

    Date1        Date2

01JAN1960    15JAN1960
02MAR1961    18FEB1962
25DEC2000    03JAN2001
01FEB2002    31MAR2002
```

Functions That Extract the Year, Month, Day, etc. from a SAS Date

This group of functions takes a SAS date value and returns parts of the date, such as the year, the month, or the day of the week. Since these functions are demonstrated in a single program, let's supply the syntax and examples.

Function: YEAR

Purpose: To extract the year from a SAS date.

Syntax: **YEAR(*date*)**

date is a SAS date value.

Examples

Function	Returns
YEAR('16AUG2002'd)	2002
YEAR('16AUG02'd)	2002

See Program 4.5.

Function: QTR

Purpose: To extract the quarter (January–March = 1, April–June = 2, etc.) from a SAS date.

Syntax: **QTR(*date*)**

date is a SAS date value.

Examples

Function	Returns
QTR('05FEB2003'd)	1
QTR('01DEC2003'd)	4

See Program 4.5.

Function: MONTH

Purpose: To extract the month of the year from a SAS date (1 = January, 2=February, etc.).

Syntax: **MONTH(*date*)**

date is a SAS date value.

Examples

Function	Returns
MONTH('16AUG2002'd)	8

See Program 4.5.

Function: WEEK

Purpose: To extract the week number of the year from a SAS date (the week-number value is a number from 0 to 53 or 1 to 53, depending on the optional modifier).

Syntax: **WEEK(<*date*> <,'*modifier*'>))**

date is a SAS date value. If *date* is omitted, the WEEK function returns the week number of the current date.

modifier is an optional argument that determines how the week-number value is determined. If *modifier* is omitted, the first Sunday of the year is week 1. For dates prior to this date, the WEEK function returns a 0. The various modifiers provide several different methods for computing the value returned by the WEEK function. Most users will probably want to use this function without any modifiers. For details about the modifiers, see Product Documentation in the Knowledge Base, available at http://support.sas.com/documentation.

Examples

Function	Returns
`WEEK('16AUG2002'd)`	`32`
`WEEK('01JAN1960'd)`	`0`
`WEEK('03JAN1960'd)`	`1`
`WEEK('01JAN1960'd,'V')`	`53`

See Program 4.5 for an example.

Function: WEEKDAY

Purpose: To extract the day of the week from a SAS date (1 = Sunday, 2=Monday, etc.).

Syntax: **WEEKDAY(*date*)**

date is a SAS date value.

Examples

Function	Returns
WEEKDAY('16AUG2002'd)	5 (Thursday)

Function: DAY

Purpose: To extract the day of the month from a SAS date, a number from 1 to 31.

Syntax: **DAY(*date*)**

date is a SAS date value.

Examples

Function	Returns
DAY('16AUG2002'd)	16

See Program 4.5.

Program 4.5: Demonstrating the functions YEAR, QTR, MONTH, WEEK, DAY, and WEEKDAY

```
***Primary functions: YEAR, QTR, MONTH, WEEK, DAY, and WEEKDAY;

data date_functions;
   set dates(drop=Date2);
   Year = year(Date1);
   Quarter = qtr(Date1);
   Month = month(Date1);
   Week = week(Date1);
   Day_of_month = day(Date1);
```

```
   Day_of_week = weekday(Date1);
run;
title "Listing of Data Set DATE_FUNCTIONS";
proc print data=date_functions noobs;
run;
```

Explanation

These basic date functions are straightforward. They all take a SAS date as the single argument and return the year, the quarter, the month, the week, the day of the month, or the day of the week. Remember that the WEEKDAY function returns the day of the **week**, while the DAY function returns the day of the **month** (it's easy to confuse these two functions). A listing of DATE_FUNCTIONS follows:

```
Listing of Data Set DATE_FUNCTIONS

                                                        Day_of_    Day_of_
    Date1     Year     Quarter     Month     Week       month       week

01JAN1960     1960         1           1         0          1          6
02MAR1961     1961         1           3         9          2          5
25DEC2000     2000         4          12        52         25          2
01FEB2002     2002         1           2         4          1          6
```

Functions That Extract Hours, Minutes, and Seconds from SAS Datetime and Time Values

The HOUR, MINUTE, and SECOND functions work with SAS datetime or time values in much the same way as the MONTH, YEAR, and WEEKDAY functions work with SAS date values.

Function: HOUR

Purpose: To extract the hour from a SAS datetime or time value.

Syntax: **HOUR(*time* or *dt*)**

time or *dt* is a SAS time or datetime value.

Examples

For these examples, `DT = '02JAN1960:5:10:15'dt, T = '5:8:10'T.`

Function	Returns
HOUR(DT)	5
HOUR(T)	5
HOUR(HMS(5,8,9))	5

See Program 4.6.

Function: MINUTE

Purpose: To extract the minute value from a SAS datetime or time value.

Syntax: **MINUTE(*time* or *dt*)**

time or *dt* is a SAS time or datetime value.

Examples

For these examples, `DT = '02JAN1960:5:10:15'dt, T = '5:8:10'T.`

Function	Returns
MINUTE(DT)	5
MINUTE(T)	5
MINUTE(HMS(5,8,9))	5

See Program 4.6.

Function: SECOND

Purpose: To extract the second value from a SAS datetime or time value.

Syntax: **SECOND(*time* or *dt*)**

time or *dt* is a SAS time or datetime value.

Examples

For these examples, `DT = '02JAN1960:5:10:15'dt, T = '5:8:10'T.`

Function	Returns
SECOND(DT)	15
SECOND(T)	10
SECOND(HMS(5,8,9))	9

Program 4.6: Demonstrating the HOUR, MINUTE, and SECOND functions

```
***Primary functions: HOUR, MINUTE, and SECOND;

data time;
   DT = '01jan1960:5:15:30'dt;
   T = '10:05:23't;
   Hour_dt = hour(DT);
   Hour_time = hour(T);
   Minute_dt = minute(DT);
   Minute_time = minute(T);
   Second_dt = second(DT);
   Second_time = second(T);
   format DT datetime.;
run;

title "Listing of Data Set TIME";
proc print data=time noobs heading=h;
run;
```

Explanation

The variable DT is a SAS datetime value (computed as a SAS datetime constant), and T is a SAS time value (computed as a SAS time constant). The program demonstrates that the HOUR, MINUTE, and SECOND functions can take either SAS datetime or time values as arguments. The listing follows:

```
Listing of Data Set TIME

                                Hour_ Minute_ Minute_ Second_ Second_
      DT           T   Hour_dt  time    dt     time     dt     time

01JAN60:05:15:30 36323    5      10     15      5       30      23
```

Functions That Extract the Date or Time from SAS Datetime Values

The DATEPART and TIMEPART functions extract either the date or the time from a SAS datetime value (the number of seconds from January 1, 1960).

Function: DATEPART

Purpose: To compute a SAS date from a SAS datetime value.

Syntax: **DATEPART(*date-time-value*)**

date-time-value is a SAS datetime value.

Examples

For these examples, `DT = '02JAN1960:5:10:15'dt.`

Function	Returns
`DATEPART(DT)`	`1` (`01JAN1960` – formatted)
`DATEPART('4JUN2003:20:48:15'DT)`	`15860` (`04JUN2003` – formatted)

See Program 4.7.

Function: TIMEPART

Purpose: To extract the time part of a SAS datetime value.

Syntax: **TIMEPART(*date-time-value*)**

Date-time-value is a SAS datetime value.

Examples

For these examples, `DT = '02JAN1960:5:10:15'dt.`

Function	Returns
`TIMEPART(DT)`	`18615 (5:10:15 – formatted)`
`TIMEPART('4JUN2003:20:48:15'DT)`	`74895 (20:48:15 – formatted)`

Program 4.7: Extracting the date part and time part of a SAS datetime value

```
   ***Primary functions: DATEPART and TIMEPART;

data pieces_parts;
   DT = '01jan1960:5:15:30'dt;
   Date = datepart(DT);
   Time = timepart(DT);
   format DT datetime. Time time. Date date9.;
run;

title "Listing of Data Set PIECES_PARTS";
proc print data=pieces_parts noobs;
run;
```

Explanation

The DATEPART and TIMEPART functions extract the date and the time from the datetime value, respectively. These two functions are especially useful when you import data from other sources. (In SAS 8, imported spreadsheet columns that were formatted as dates in Microsoft Excel wound up as datetime values in the SAS data set.) You can use these two functions to separate the date and time from that value. See the following listing:

```
Listing of Data Set PIECES_PARTS

       DT                Date     Time
01JAN60:05:15:30    01JAN1960    5:15:30
```

Functions That Work with Date, Datetime, and Time Intervals

Functions in this group work with date or time intervals. The INTCK function, when used with date or datetime values, can determine the number of interval boundaries crossed between two dates. When used with SAS time values, it can determine the number of hour, minute, or second boundaries between two time values.

The INTNX function, when used with SAS date or datetime values, is used to determine the date after a given number of intervals have passed. When used with SAS time values, it computes the time after a given number of time interval units have passed.

You will find an excellent description of these two functions in *SAS Language Reference: Concepts* or in the following technical note:

`http://support.sas.com/techsup/technote/ts668.html`

Note: The wording of arguments in this book might differ from the wording of arguments in the Product Documentation at http://support.sas.com.

Function: INTCK

Purpose: To return the number of intervals between two dates, two times, or two datetime values. To be more accurate, the INTCK function counts the number of times a boundary has been crossed going from the first value to the second.

For example, if the interval is YEAR and the starting date is January 1, 2002, and the ending date is December 31, 2002, the function returns a 0. The reason for this is that the boundary for YEAR is January 1, and even

though the starting date is on a boundary, no boundaries are crossed in going from the first date to the second. Using the same logic, going from December 31, 2002, to January 1, 2003, *does* cross a year boundary and returns a 1. This is true even though, in the first case, there are 364 days between the dates and, in the latter case, only one day.

These intervals can be used "as is" or with multipliers such as two-year intervals, and they can be shifted so that the boundary is, for example, the seventh month of the year (July) instead of January 1.

When used with multi-intervals and shifted intervals, the INTCK function can become very complicated. A limited discussion of the finer points of the INTCK function follows the syntax and examples.

Syntax: **INTCK('*interval<Multiple><.shift>*', *start-value*, *end-value*)**

Intervals can be date units:

Interval	Description
DAY	Day
WEEK	Week
WEEKDAY	Each weekday (Monday to Friday, or any set of days you choose)
TENDAY	Ten-day period
SEMIMONTH	Two-week period
MONTH	Month
QTR	Quarter (Jan–Mar = 1, Apr–Jun = 2, etc.)
SEMIYEAR	Half year
YEAR	Year

Intervals can be time units:

Interval	Description
SECOND	Seconds
MINUTE	Minutes
HOUR	Hours

Intervals can be datetime units:

Interval	Description
DTDAY	Day
DTWEEK	Week
DTWEEKDAY	Each weekday (Monday to Friday)
DTTENDAY	Ten-day period
DTSEMIMONTH	Two-week period
DTMONTH	Month
DTQTR	Quarter (Jan–Mar = 1, Apr–Jun = 2, etc.)
DTSEMIYEAR	Half year
DTYEAR	Year

interval is one item from the preceding list, placed in quotation marks.

multiple is an optional modifier in the interval. You can specify multiples of an interval. For example, MONTH2 specifies two-month intervals; DAY50 specifies 50-day intervals.

.shift is an optional parameter that determines the starting point in an interval. For example, YEAR.4 specifies yearly intervals, starting from April 1. The shift value for single intervals is shown in the following table:

Shift value for SAS date and datetime values:

Interval	Shift Value
YEAR	Month
SEMIYEAR	Month
QTR	Month
MONTH	Month
SEMIMONTH	Semimonth*
TENDAY	Tenday
WEEKDAY	Day
WEEK	Day
DAY	Day

Shift value for SAS time intervals:

Interval	Shift Value
HOUR	Hour*
MINUTE	Minute*
SECOND	Second*

*Only multi-intervals of these intervals can be shifted.

For all multi-unit intervals except WEEK, SAS creates an interval starting from January 1, 1960. Multiple intervals are all shifted by the same unit as the non-multiple intervals (see lists above). So, YEAR4.24 specifies four-year intervals with the interval boundary at the beginning of the second year (January 1, 1962, January 1, 1966, etc.). MONTH4.2 indicates four-month intervals, with the boundary being the first day of the second month. See the following discussion on interval multipliers and shifted intervals.

Here are some examples of intervals:

Interval	Interpretation
YEAR	Each year
YEAR2	Every two years
YEAR.4	Each April
YEAR4.11	November, every four years
MONTH	Every month
MONTH4	Every four months
MONTH6.3	Every six months with boundaries at March and September
WEEK	Each week
WEEK2	Every two weeks
WEEK.4	Every week starting with Wednesday
WEEK2.4	Every two weeks starting with Wednesday
WEEKDAY	Five-day weeks with weekend days, Saturday and Sunday
WEEKDAY1W	Six-day weeks with weekend day, Sunday
WEEKDAY12W	Five-day weeks with weekend days, Sunday and Monday
HOUR	Every hour
HOUR4	Every four hours
HOUR8.7	Every eight hours with boundaries 6 AM, 2 PM, and 10 PM
DTMONTH	Every month (used with datetime values)

start-value is a SAS date, time, or datetime value.

end-value is a SAS date, time, or datetime value.

Examples

Function	Returns
`INTCK('WEEK','16AUG2002'd,'24AUG2002'd)`	1
`INTCK('YEAR', '01JAN2002'd,'31DEC2002'd)`	0
`INTCK('YEAR', '01JAN2002'd,'02JAN2003'd)`	1
`INTCK('YEAR', '31DEC2002'd,'01JAN2003'd)`	1
`INTCK('QTR','01JAN2002'd,'01AUG2002'd)`	2
`INTCK('MONTH3','01JAN2002'd,'15APR2002'd)`	1
`INTCK('YEAR.7','05MAY2002'd,'15JUL2002'd)`	1
`INTCK('HOUR','06:01:00't,'07:23:15't)`	1

See Program 4.9.

A Discussion of Interval Multipliers and Shifted Intervals

Some applications of interval multipliers are quite straightforward. For example, if you use YEAR2 as your interval, the intervals will be every two years. The value of

```
INTCK('YEAR2','15JAN2000'd,'21JAN2003'd)
```

is equal to 1 (one boundary, January 1, 2002, was crossed in going from January 15, 2000, to January 21, 2002). The reason that January 1, 2002, is a boundary is that the counting of boundaries goes back to January 1, 1960, which was an even number. Therefore, the boundaries will be even-numbered years.

You can shift some single intervals. For example, YEAR.7 indicates yearly intervals with the boundary being July 1 of every year. For the intervals of YEAR, SEMIYEAR, and QTR, the shift amount is months. For example, the value of

```
INTCK('YEAR.7','01JUN2000'd,'03JUL2002'd)
```

is equal to 3 (crossing boundaries at July 1, 2000, July 1, 2001, and July 1, 2002).

Shifting intervals that use multipliers is similar. For example, YEAR2.12 indicates two-year intervals, with boundaries at the second year of each interval: January 1, 1961, January 1, 1963, etc. That is, every odd year. For example, the value of

```
INTCK('YEAR2.12','15JAN2000'd,'21JAN2003'd)
```

is equal to 2 (crossing the boundaries at January 1, 2001, and January 1, 2003).

Multi-month intervals are shifted by months, not weeks (since there is not an even number of weeks in a month). MONTH4.2 means four-month intervals with the boundary being the second month of each four-month period. By the way, the .2 does not mean "shift the boundary by 2 months." It means the boundary is the second month of each interval. As Charley Mullin says in his technical note: "The boundary is shifted TO an interval, not BY the interval."

The value of

```
INTCK('MONTH4.2','28JAN2003'd,'03JUL2003'd)
```

is equal to 2 (crossing the boundaries at February 1, 2003, and June 1, 2003).

WEEK and multi-week intervals present a special problem. For example, you might expect the value of

```
INTCK(WEEK,'01JAN1960'd,'04JAN1960'd)
```

to equal 0. However, it is equal to 1. The problem is that weekly intervals are counted every time a Sunday is crossed and January 1, 1960, is a Friday. The way that SAS decided to solve this problem was to start counting from Sunday in the same week of January 1, 1960, which is December 27, 1959. Going from January 1, 1960, to January 4, 1960, crosses a boundary (Sunday, January 3). This gets even more complicated when you are dealing with multi-week intervals.

As the default, the interval of WEEKDAY treats Saturday and Sunday as part of the preceding day. For example, the value of

```
INTCK('WEEKDAY','01JUN2003'd,'30JUN2003'd)
```

is equal to 21. June 1, 2003, is a Sunday, and June 30 falls on a Monday. The number of times you have crossed a boundary (a working day) is 21. (Please see a further discussion of the WEEKDAY interval below the June 2009 calendar later in this chapter.)

You can specify days other than Saturday and Sunday to be treated as weekend days. For example, if you had a six-day work week, with Sunday as the day off, you could indicate the interval as WEEKDAY1W. So, the value of

```
INTCK('WEEKDAY1W','01JUN2003'd,'30JUN2003'd)
```

is equal to 25 (Monday through Saturday for four weeks plus Monday, June 30).

If you were in the restaurant business and your restaurant was closed on Sunday and Monday, you would use the interval: WEEKDAY12W to compute the number of work days between two dates.

When you are computing the number of working days between two dates, it is important to know if the starting date is a working day or not. For example, take a look at the following calendar for June 2009:

JUNE

S	M	T	W	T	F	S
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

The expression `INTCK('weekday','08jun2009'd,'12jun2009'd)` returns a 4, even though there are 5 working days in the week. Remember, you are counting how many boundaries are being crossed going from date 1 to date 2. Here, you cross a boundary on June 9, 10, 11, and 12 (4 boundaries).

Now, see what happens if you start on June 7 instead of June 8. The expression `INTCK('weekday','07jun2009'd,'12jun2009'd)` returns a 5, since you cross one boundary going from Sunday to Monday. By the way, if you computed the interval from June 6 to June 12, the result would also be a 5.

Notice that it doesn't matter if the ending date is a weekday or not. For example, `INTCK('weekday','08jun2009'd,'13jun2009'd)` or `INTCK('weekday','08jun2009'd,'14jun2009'd)` still returns a value of 4.

Function: INTNX

Purpose: To return the date after a specified number of intervals have passed.

Syntax: **INTNX('*interval*', *start-date*, *increment* <,'*alignment*'>)**

interval is one of the same values that are used with the INTCK function (placed in quotation marks).

start-date is a SAS date.

increment is the number of intervals between the start date and the date returned by the function.

alignment is an optional argument and has a value of BEGINNING (B), MIDDLE (M), END (E), or SAMEDAY(S). The default is BEGINNING. For example, if the interval is WEEK, an increment of 1 from January 1, 1960, with the default returns the date January 3, 1960 (a Sunday, the beginning of a boundary). The same date and interval with an alignment of MIDDLE returns the date January 6, 1960 (a Wednesday, the middle of the interval).

Examples

For these examples, `DT1 = '01JAN1960:7:5:12'DT.`
Note: Values in parentheses in the Returns column are the formatted values.

Function	Returns
INTNX('WEEK','01JAN1960'd,1)	2 (Sunday, Jan 3, 1960)
INTNX('WEEK','01JAN1960'd,1,'MIDDLE')	5 (Wednesday, Jan 6, 1960)
INTNX('WEEK.4','01JAN1960'd,1)	5 (Wednesday, Jan 6, 1960)
INTNX('WEEK2','01JAN1960'd,1)	9 (Sunday, Jan 10, 1960)
INTNX('QTR','01JAN2003'd,1)	15796 (Tuesday, April 1, 2003)
INTNX('YEAR.3','01JAN2003'd,1)	15765 (Saturday, March 1, 2003)
INTNX('YEAR.3','01JAN2003'd,2)	16131 (Monday, March 1, 2004)
INTNX('YEAR','01JUN2003'd,1)	16071 (Thursday, January 1, 2004)
INTNX('YEAR','01JUN2003'd,2)	16437 (Saturday, January 1, 2005)
INTNX('YEAR4.11','01JAN2003'd,1)	16376 (Monday, November 1, 2004)
INTNX('DTMONTH',DT1,3)	7862400 (01APR60:00:00:00)
INTNX('HOUR','9:15:09'T,2)	39600 (11:00:00)
INTNX('YEAR','15JAN1960'D,-1)	-365 (January 1, 1959)

Some examples demonstrating the SAMEDAY alignment

Date = '10May2005'd (Tuesday). Return values are formatted.

Function	Returns
INTNX('week',Date,1,'sameday')	17May2005 (Tuesday)
INTNX('month',Date,1,'sameday')	10Jun2005 (Friday)
INTNX('year',Date1,1,'sameday')	10May2006 (Wednesday)
INTNX('weekday','7May2005'd,1, 'sameday') Note: this is a Saturday	09May2005 (Monday)

Program 4.8: Demonstrating the INTNX function (with the SAMEDAY alignment)

```
***Primary functions: INTNX, WEEKDAY;
***Other functions: RANUNI, CEIL;

*A dentist wants to see each of his patients in six months for a followup
visit.  However, if the date in six months falls on a Saturday or Sunday,
he wants to pick a random day in the following week.;

Data dental;
   input Patno : $5. Visit_date : mmddyy10.;
   format Visit_date weekdate.;
datalines;
001 1/14/2009
002 1/17/2009
003 1/18/2009
004 1/19/2009
005 1/19/2009
006 1/20/2009
007 1/11/2009
008 1/17/2009
;
title "Listing of data set DENTAL";
proc print data=dental noobs;
run;
data followup;
   set dental;
   Six_months = intnx('month',Visit_date,6,'sameday');
   *Check if weekend;
   DayofWeek = weekday(six_months);
   *Keep track of actual day for testing purposes;
   Actual = Six_months;
   *If Sunday add random integer between 1 and 5;
   if DayofWeek = 1 then
      Six_months = Six_months + ceil(ranuni(0)*5);
   *If Saturday, add a random integer between 2 and 6;
   else if DayofWeek = 7 then
      Six_months = Six_months + ceil(ranuni(0)*5 + 1);
run;
title "Six Month Appointment Dates";
proc report data=followup nowd headline;
   columns Patno Visit_date Actual Six_months;
   define Patno / display "Patient Number" width=7;
   define Visit_date / display "Initial Date" width=15 format=weekdate.;
   define Actual / display "Actual Day" width=15 format=weekdate.;
   define Six_months / display "Six Month Appt." width=15
format=weekdate.;
run;
quit;
```

Explanation

The introduction of the SAMEDAY alignment greatly enhanced the usefulness of the INTNX function. If you used the INTNX function in the preceding program without the SAMEDAY alignment, all of the dentist's patients would be coming in on the first of each month! Not a great plan. By using the SAMEDAY alignment, the function returns a date six months ahead, but on the same day of the month. Since this date may be a Saturday or Sunday, adjustments need to be made. In this program, it was decided that if the six month date fell on a Saturday or Sunday, a random day in the following week was to be chosen.

The expression `ceil(ranuni(0)*5)` produces a random integer from 1 to 5; the expression `ceil(ranuni(0)*5 + 1)` produces a random integer from 2 to 6. For illustration purposes, the actual date six months from the visit date was not dropped from the data set so that you can see what happens if the follow-up date falls on a Saturday or Sunday. Here is the listing:

```
Six Month Appointment Dates

  Patient
  Number      Initial Date       Actual Day  Six Month Appt.
  ____________________________________________________________
  001      Wed, Jan 14, 09  Tue, Jul 14, 09  Tue, Jul 14, 09
  002      Sat, Jan 17, 09  Fri, Jul 17, 09  Fri, Jul 17, 09
  003      Sun, Jan 18, 09  Sat, Jul 18, 09  Tue, Jul 21, 09
  004      Mon, Jan 19, 09  Sun, Jul 19, 09  Wed, Jul 22, 09
  005      Mon, Jan 19, 09  Sun, Jul 19, 09  Fri, Jul 24, 09
  006      Tue, Jan 20, 09  Mon, Jul 20, 09  Mon, Jul 20, 09
  007      Sun, Jan 11, 09  Sat, Jul 11, 09  Mon, Jul 13, 09
  008      Sat, Jan 17, 09  Fri, Jul 17, 09  Fri, Jul 17, 09
```

Using the INTNX Function to Determine Starting Boundaries for Multi-Day Intervals

Interval boundaries are straightforward for intervals such as years, quarters, and months. However, suppose you want to create 12-day intervals. How many 12-day intervals are there from January 1, 2004, to January 11, 2004? How many boundaries have you crossed? The

problem here is that you have to realize that you start counting 12-day intervals from January 1, 1960, to determine where the boundaries are. Here's an easy way to see what date the counting starts on: use the INTNX function like this:

```
START_INTERVAL = INTNX('DAY12','01JAN2004'd,1)
```

The value is Saturday, January 10, 2004. So, in going from January 1, 2004, to January 11, 2004, you cross one boundary (January 10, 2004). To check, note that

```
INTCK('DAY12','01JAN2004'd,'11JAN2004'd)
```

is equal to 1.

Function: YRDIF

Purpose: To return the difference in years between two dates (includes fractional parts of a year).

Important note: About the time this book was being sent to the printer, it was discovered that the YRDIF function would sometimes return a value that was off by one day for certain date intervals. It appeared this error was related to leap years. However, this author believes that, even with this error, using the YRDIF function to compute ages (or any differences in years) is still more accurate than the older method of dividing the difference in years by 365.25. Future releases of SAS are expected to address this error with YRDIF. If you need to compute exact year differences and you are using a version of SAS that does not have the updated feature, you can use the INTCK function to accomplish your goals (see an illustration in this section).

Syntax: **YRDIF(*start-date*, *end-date*, '*basis*')**

start-date is a SAS date value.

end-date is a SAS date value.

basis is an argument that controls how SAS computes the result. The first value is used to specify the number of days in a month; the second value (after the slash) is used to specify the number of days in a year.

A value of 'ACT/ACT' (alias 'ACTUAL') uses the actual number of days in a month and the actual number of days in a year (either 365 or 366 days, depending on whether there are leap years involved). For certain industries, especially financial institutions, you can specify values for the number of days in the month and the number of days in the year. This is frequently done for interest calculations on bonds and other commodities. Other choices for *basis* are:

'30/360'	Uses 30-day months and 360-day years in the calculation.
'ACT/365'	Uses the actual number of days between the two dates, but uses 365-day years, even if a leap year is in the interval.
'ACT/360'	Uses the actual number of days between the two dates, but uses 360-day years.

Examples

Function	Returns
YRDIF('01JAN2002'd,'01JAN2003'd,'ACTUAL')	1
YRDIF('01JAN2002'd,01FEB2002'd,'ACT/ACT')	.0849
YRDIF('01FEB2002'd,01MAR2003'd,'ACTUAL')	1.9767
YRDIF('01JAN2002'd,'01JAN2003'd,'ACT/365')	1.0139

Program 4.9: Program to demonstrate the date interval functions

```
***Primary functions: INTCK, INTNX, YRDIF;

data period;
   set dates;
   Interval_month = intck('month',Date1,Date2);
   Interval_year  = intck('year',Date1,Date2);
   Year_diff      = yrdif(Date1,Date2,'actual');
   Interval_qtr   = intck('qtr',Date1,Date2);
   Next_month     = intnx('month',Date1,1);
   Next_year      = intnx('year',Date1,1);
   Next_qtr       = intnx('qtr',Date1,1);
   Six_month      = intnx('month',Date1,6);
   format Next: Six_month date9.;
run;
```

```
title "Listing of Data Set PERIOD";
proc print data=period heading=h;
   id date1 date2;
run;
```

Explanation

Before we discuss the date functions in this program, let me point out that the ID statement of PROC PRINT lists both DATE1 and DATE2 as ID variables. This allows the values to be repeated on the lower portion of the listing.

The interval functions can be somewhat confusing. It helps to keep in mind that the INTCK function counts how many times you cross a boundary going from the start date to the end date. The listing follows:

```
Listing of Data Set PERIOD

                   Interval_ Interval_  Year_  Interval_     Next_
    Date1     Date2   month      year     diff     qtr        month

01JAN1960 15JAN1960      0         0     0.03825     0     01FEB1960
02MAR1961 18FEB1962     11         1     0.96712     4     01APR1961
25DEC2000 03JAN2001      1         1     0.02461     1     01JAN2001
01FEB2002 31MAR2002      1         0     0.15890     0     01MAR2002


    Date1     Date2 Next_year     Next_qtr    Six_month

01JAN1960 15JAN1960 01JAN1961    01APR1960    01JUL1960
02MAR1961 18FEB1962 01JAN1962    01APR1961    01SEP1961
25DEC2000 03JAN2001 01JAN2001    01JAN2001    01JUN2001
01FEB2002 31MAR2002 01JAN2003    01APR2002    01AUG2002
```

Computing Exact Ages

If you are using a release of SAS that has not corrected the possible error in the YRDIF function, you can use the following SAS statement to compute AGE exactly (submitted by my friend Mike Zdeb):

```
Age_exact = floor((intck('month',DOB,Date)-(day(Date) < day(DOB))) / 12);
```

Function That Computes Dates of Standard Holidays

Function: HOLIDAY

Purpose: Returns a SAS date, given a holiday name and a year.

Syntax: **HOLIDAY (*holiday*, *year*)**

holiday is a holiday name (see list below).

year is a numeric variable or constant that represents the year.

Partial List of Holidays:

Christmas	Christmas day (December 25)
Columbus	Columbus day (2nd Monday in October)
Easter	Easter Sunday
Fathers	Father's Day (3rd Sunday in June)
Halloween	Halloween
Labor	Labor Day (1st Monday in September)
MLK	Martin Luther King Day (celebrated on Monday)
Memorial	Memorial Day (1st Monday in May)
Mothers	Mother's Day (2nd Sunday in May)
Newyear	New Year's Day (January 1)
Thanksgiving	Thanksgiving (4th Thursday in November)
USIndependence	July 4th holiday
USPresidents	President's Day (3rd Monday in February)
Veterans	Veterans Day (November 11)
VeteransUSG	Veterans Day (U.S. Government)

Examples

Function	Returns
`HOLIDAY('Christmas',2009)`	`12/25/2009 (Friday)`
`HOLIDAY('USIndependence',2009)`	`7/4/2009 (Saturday)`
`HOLIDAY('VeteransUSG',2009)`	`11/11/2009 (Wednesday)`
`HOLIDAY('Easter',2009)`	`4/12/2009 (Sunday)`
`HOLIDAY('MLK',2009)`	`1/19/2009 (Monday)`
`HOLIDAY('Thanksgiving',2009)`	`11/26/2009 (Thursday)`

Program 4.10: Demonstrating the HOLIDAY function

```
***Primary function: HOLIDAY;
***Other functions: WEEKDAY, INTCK;
data salary;
   H1 = holiday('Newyear',2005);
   if weekday(H1) = 7 then H1 = H1 + 2;
   else if weekday(H1) = 1 then H1 = H1 + 1;
   H2 = holiday('MLK',2005);
   H3 = holiday('USpresidents',2005);
   H4 = holiday('Easter',2005)-2;
   array H[4];
   First = '01Jan2005'd; *Saturday;
   Second = '31Mar2005'd; *Thursday;
   Work = intck('weekday',First,Second);
   /* if holiday falls between the First and Second date,
      decrement number of working days */
   do i = 1 to 4;
      if First le H[i] le Second then Work = Work - 1;
   end;
   Salary = 500 * Work;
   format First Second mmddyy10. Salary dollar10.;
   keep First Second Work Salary;
run;
title "Listing of SALARY";
proc print data=SALARY noobs;
run;
```

Explanation

In this program, you want to compute the number of working days between January 1, 2005 (Saturday) and March 31, 2005.

The INTCK function with the WEEKDAY interval computes the number of times you cross working day boundaries going from one date to another (with Monday through Friday being defined as the default working days).

Note: Since the starting date is a Saturday, you do not have to add one to the value returned, because going from a weekend day to Monday crosses a boundary. If the starting date were not on a weekend, you would need to add one to the variable WORK. (Please see the explanation following the June 2009 calendar earlier in this chapter.)

Next, you want to test if any of the holidays (New Year's Day, Martin Luther King's birthday, President's Day, or Easter) fall in that interval. In addition, since New Year's Day can fall on any day of the week, you use the WEEKDAY function to test if this holiday falls on a Saturday or Sunday. If so, your company gives its employees the following Monday off. Easter always falls on a Sunday, so employees are given the previous Friday (Good Friday) off.

An array is created to hold the four non-working days. Finally, you test if each of the four non-working days fall in the given interval. If so, you decrement the number of working days computed by the INTCK function.

Here is the listing:

```
Listing of SALARY

     First         Second     Work         Salary

01/01/2005     03/31/2005      60         $30,000
```

Functions That Work with Julian Dates

This group of functions involves Julian dates. Julian dates are commonly used in computer applications and represent a date as a two- or four-digit year followed by a three-digit day of the year (1 to 365 or 366, if it is a leap year). For example, January 3, 2003, in Julian notation would be either 2003003 or 03003. December 31, 2003 (a non-leap year) would be either 2003365 or 03365.

Function: DATEJUL

Purpose: To convert a Julian date into a SAS date.

Syntax: **DATEJUL(*jul-date*)**

jul-date is a numerical value representing the Julian date in the form *dddyy* or *dddyyyy*.

Examples

For these examples, `JDATE = 1960123.`

Function	Returns
`DATEJUL(1960001)`	`0` (01JAN1960 formatted)
`DATEJUL(2003365)`	`16070` (31DEC2003 formatted)
`DATEJUL(JDATE)`	`122` (02MAY1960 formatted)

See Program 4.11.

Function: JULDATE

Purpose: To convert a SAS date into a Julian date.

Syntax: **`JULDATE(`*`date`*`)`**

`date` is a SAS date.

Examples

For these examples, `DATE = '31DEC2003'D.`

Function	Returns
`JULDATE(DATE)`	`3365`
`JULDATE('01JAN1960'D)`	`60001`
`JULDATE(122)`	`60123`

See Program 4.11.

Function: JULDATE7

Purpose: To convert a SAS date into seven-digit Julian date.

Syntax: **JULDATE7(*date*)**

date is a SAS date.

Examples

For these examples, `DATE = '31DEC2003'D.`

Function	Returns
JULDATE7(DATE)	2003365
JULDATE7('01JAN1960'D)	1960001
JULDATE7(122)	1960123

Program 4.11: Demonstrating the three Julian date functions

```
   ***Primary functions: DATEJUL, JULDATE, and JULDATE7.;

***Note: option YEARCUTOFF set to 1920;
options yearcutoff = 1920;
data julian;
   input Date : date9. Jdate;
   Jdate_to_sas = datejul(Jdate);
   Sas_to_Jdate = juldate(Date);
   Sas_to_jdate7 = juldate7(Date);
   format Date Jdate_to_sas mmddyy10.;
datalines;
01JAN1960 2003365
15MAY1901 1905001
21OCT1946 5001
;

title "Listing of Data Set JULIAN";
proc print data=julian noobs;
   var Date Sas_to_jdate Sas_to_jdate7 Jdate Jdate_to_sas;
run;
```

Explanation

It is important to realize that Julian dates without four-digit years will be converted to SAS dates, based on the value of the YEARCUTOFF system option. To avoid any problems, it is best to use seven-digit Julian dates. The listing is shown next:

```
Listing of Data Set JULIAN

              Sas_to_     Sas_to_                    Jdate_
      Date     Jdate       jdate7       Jdate        to_sas

01/01/1960      60001     1960001     2003365     12/31/2003
05/15/1901    1901135     1901135     1905001     01/01/1905
10/21/1946      46294     1946294        5001     01/01/2005
```

Index

D

G

H

I

M

N

Q

R

S