Harnessing the Power of SAS ISO 8601 Informats, Formats, and the CALL IS8601_CONVERT Routine

Kim Wilson, SAS Institute Inc., Cary, NC, USA

ABSTRACT

Clinical Data Interchange Standards Consortium (CDISC) is a data standards group that governs clinical research around the world. This data consists of many date, time, datetime, duration, and interval values that must be expressed in a consistent manner across many organizations. The International Organization for Standardization (ISO) approved the ISO 8601 standard for representing dates and times, and this standard is compliant with CDISC.

This paper addresses how to create and manage ISO 8601 compliant date, time, and datetime values in a CDISC environment. The paper also discusses the computation of durations and intervals. Examples that use the SAS call routine CALL IS8601_CONVERT and other programming logic are also provided, along with helpful tips and suggestions. In addition, the paper presents solutions to some common date and time problems, such as handling missing date components.

INTRODUCTION

ISO 8601 is an international standard for representing dates and time, including many variations for representing dates, times, and intervals. These representations allow values in a basic format, meaning no delimiters between components, or in an extended format, which includes delimiters between components. SAS introduced the ISO 8601 family of informats and formats beginning in SAS® 8.2 and first documented them in the *SAS 9.1 XML LIBNAME Engine: User's Guide*. The original names for the delimited informats and formats contained the prefix IS8601 while those without delimiters contained the prefix ND8601. Beginning in SAS 9.2, the names changed so that informats and formats having a prefix of ND8601 became B8601 (B for basic) while those with the prefix IS8601 changed to E8601 (E for extended).

During clinical trials, these informats and formats are helpful when you are reading and writing data into and out of SAS and calculating durations and intervals relating to events that are recorded in the study.

ISO 8601 REPRESENTATION

The two main representations of date, time, and datetime values within the ISO 8601 standards are the basic and extended notations. A value is considered *extended* when delimiters separate the various components within the value, whereas a *basic value* omits the delimiters. The extended format requires hyphen delimiters for date components and colon delimiters for time components. Spaces are not allowed in any IS0 8601 representation. The structures for each data type require that you fill each placeholder with a value, including adding a zero to single-digit months, days, hours, and minutes. When you specify a datetime value, an uppercase T is the required delimiter between the date and time. When SAS reads an ISO 8601 value that specifies a time-zone offset (+|-hh:m or +|-hhmm), the time or datetime value is adjusted to account for the offset. The computed SAS value is the time or datetime for the zero meridian, which is in Greenwich, England. Therefore, the zero meridian is called Greenwich Mean Time (GMT).

In the context of the ISO 8601 representations, the following topics are discussed:

- the structure of dates, times, and datetimes
- examples that show how to use informats and formats to read and write date, time, and datetime values to SAS date, time, and datetime variables
- durations
- intervals
- partial and missing components
- the $N8601B and $N8601E informats and formats
- the CALL IS8601_CONVERT routine

STRUCTURE FOR DATES, TIMES, AND DATETIMES

Extended informats and formats are prefixed with **E8601**, and they take these forms:

date: *yyyy-mm-dd*

time: *hh*:*mm*:*ss*<.*ffffff*>

datetime: *yyyy-mm*-d*d*T*hh*:*mm*:*ss*<.*ffffff*>

With the time-zone specification added, the formats and informats take these forms:

time: *hh*:*mm*:*ss*.<*ffffff*>+|-*hh*:*mm*

datetime: *yyyy-mm-dd*T*hh*:*mm*:*ss*<.*ffffff*>+!-*hh*:*mm*

Basic informats and formats are prefixed with **B8601**, and they take these forms:

date: *yyyymmdd*

time: *hhmmss*<*ffffff*>

datetime: *yyyymmdd*T*hhmmss*<*ffffff*> (the T delimiter remains, but other delimiters are omitted)

When a time-zone specification is included, as shown below, the time-zone difference sign remains even though the delimiters are omitted:

time: *hhmmss*<*ffffff*>+|-*hhmm*

datetime: *yyyymmdd*T*hhmmss*<*ffffff*>+|-*hhmm*

Note: Any values that are accepted by the E8601 family of informats are also accepted by the B8601 family of informats. Delimiters are not rejected by the basic informats.

The following tables list the ISO 8601 informats and formats, respectively.

Informat	Style of Value	Description
B8601CI	c*yymmddhhmmss*<*fff*>	**Reads IBM date and time.**
B8601DA/E8601DA	*yyyymmdd*	**Reads date values.**
B8601DJ	*yyyymmddhhmmss*<*fffff*>	**Reads Java date and time.**
B8601DN/E8601DN	*yyyymmdd*	**Reads date values and returns datetime value (with a time value of 000000).**
B8601DT/E8601DT	*yyyymmdd*T*hhmmss*<*fffff*>	**Reads datetime values.**
B8601DZ/E8601DZ	*yyyymmdd*T*hhmmss*+|-*hhmm*	**Reads UTC datetime values.**
B8601TM/E8601TM	*hhmmss*<*fffff*>	**Reads time values.**
B8601TZ/E8601TZ	*hhmmss*<*fffff*>+|-*hhmm*	**Reads UTC time values.**
E8601LZ	*Hh*:*mm*:*ss*+|-*hh*:*mm*.<*fffff*>	**Reads UTC time and converts to local time.**

Table 1. Basic and Extended Family of Informats

Format	Style of Value	Description
B8601DA/E8601DACI	*yyyymmdd*	**Writes date values.**
B8601DN/E8601DN	*yyyymmdd*	**Writes date values from datetime values.**
B8601DT/E8601DT	*yyyymmdd*T*hhmmss<fffff>*	**Writes datetime values.**
B8601DZ/E8601DZ	*yyyymmdd*T*hhmmss+|-hhmm*	**Writes UTC datetime values.**
B8601LZ/E8601LZ	*hhmmss+|-hhmm*	**Writes time values as local time with offset.**
B8601TM/E8601TM	*hhmmss<fffff>*	**Writes time values.**
B8601TZ/E8601TZ	*hhmmss<fffff>+|-hhmm*	**Writes time values with +0000 offset.**

Table 2. Basic and Extended Family of Formats

EXAMPLES: USING INFORMATS AND FORMATS TO READ AND WRITE DATE, TIME, AND DATETIME VALUES

The examples that follow in this section demonstrate how to use various informats to read date, time, and datetime values into SAS date, time, and datetime variables. The examples also illustrate how to use formats to write these values in a way that is meaningful to users.

Example 1: Reading Date Values

Suppose you have a clinical trial where an event begins on April 2, 2012 and ends on April 8, 2012. The dates are recorded without time values, as follows: **`20120402`** and **`2012-04-08`**. You can read these values into SAS with the B8601DA*w*. and E8601DA*w*. informats. SAS also has equivalent (like-named) formats. These formats output the newly created SAS dates in an easy-to-read layout rather than the numeric value of days since 1/1/1960.

```
data a;
   input var1 b8601da8. +1 var2 e8601da10.;
   put var1=b8601da. var2=e8601da.;
   datalines;
20120402 2012-04-08
;
run;
```

In This Example

- Because a SAS datetime value is stored as the number of seconds since January 1, 1960, the date and time portions are incorporated. Most events that are recorded during a clinical trial aim to be as complete as possible. Therefore, it is a good practice to read these values with an informat such as B8601DT*w.d* or E8601DT*w.d* so that a SAS datetime value is stored.
- The B8601DN*w*. informat reads date values and returns SAS datetime values where the time portion is **`000000`**.

Output

The resulting values for VAR1 and VAR2 are as follows:

- VAR1: **`20120402`**
- VAR2: **`2012-04-02`**

Example 2: Reading Date and Datetime Values

In the following DATA step, date, and datetime values are read into SAS with the basic and extended versions of two informats. The basic and extended versions of the formats also create SAS datetime values, which are stored as the number of seconds since January 1, 1960.

```
data a;
   input dt1 :b8601dn8. dt2 :E8601dn10. dt3 :b8601dt15. dt4 :e8601dt19.;
   put dt1=b8601dt. dt2=e8601dt. dt3=b8601dt. dt4=e8601dt. dt4=e8601dn.;
   datalines;
20120402 2012-04-02 20120402T124022 2012-04-02T12:30:22
;
run;
```

In This Example

- The variables (DT1 – DT4) are written to the SAS log using the B8601DT*w.d* and E8601DT*w.d*.formats so that all components of the date and time are shown.
- Then the variable DT4 is rewritten using the format E8601DN*w*. to show how you can output only the date portion from a value that is stored as a datetime value.

Output

The resulting values for DT1 –DT4 are as follows:

- DT1: **20120402T000000**
- D2: **2012-04-02T00:00:00**
- DT3: **20120402T124022**
- DT4: **2012-04-02T12:30:22**
- DT4 (after rewriting the value with E8601DN*w*.): **2012-04-02**

Example 3: Reading Java Styled Datetime Values

The following example reads datetime values that are output by Java:

```
data a;
   input dt1 b8601dj.;
   put dt1=b8601dt.;
   datalines;
20120402123245
;
run;
```

In This Example

- The informat B8601DJ*w*. reads datetime values without the T separator between the date and time portions. (This functionality became available in SAS 9.3.) There is no extended version of this informat because delimiters are omitted from the input values.
- The value is written to the SAS log using the B8601DT*w*. format because a B8601DJ*w*. format does not exist.

Output.

The resulting value for DT1 is **20120402T123245**.

Example 4: Reading UTC Datetime Values

Consider the following example where the offset is four hours earlier than GMT:

```
data _null_;
   x=input('2011-08-01T12:34:56-04:00',e8601dz25.);
   put x=e8601dz25.;
run;
```

In This Example

- The B8601DZ*w.d* and E8601DZ*w.d* informats read Coordinated Universal Time (UTC) datetime values that contain the datetime components along with a time-zone offset specification. The provided offset creates a SAS datetime value that is adjusted by the proper number of hours from GMT. The E8601DZ*w.d* informat converts the datetime value to GMT so that it becomes **16:34:56**.
- When the E8601DZ*w*. format displays the value, it shows the time with a **+00:00** offset.

Pointer

Remember that whenever the B8601DZ*w*. and E8601DZ*w*. formats are specified to output a datetime value, the value is assumed to be a GMT datetime value. This means the time-zone offset is always **+00:00** in the output.

Output

The resulting value for X is **2011-08-01T16:34:56+00:00**.

Example 5: Reading Time Values That Contain Time-Zone Offsets

The next example demonstrates how to read time values with time-zone offsets in order to create GMT time values.

```
data _null_;
   x=input('12:34:56-04:00',e8601tz14.);
   put x=e8601tz14.;
   put x=b8601tz.;
run;
```

In This Example

- The B8601TZ*w.d* and E8601TZ*w.d* informats read time values along with time-zone offsets in order to create GMT time values.
- The E8601TZ*w*. format writes the SAS time value with the time-zone offset as **+00:00**.
- The B8601TZ*w*. format writes the SAS time value with the time-zone offset as **+0000**.

Note: When SAS reads a UTC time by using the B8601TZ*w.d* informat and the adjusted time is greater than 24 hours or less than 00 hours, SAS adjusts the value so that the time is between 0 and 23:59:59 (one second before midnight).

Output

The resulting values for X are as follows:

- **16:34:56+00:00**
- **163456+0000**

Example 6: Writing Local Times That Include Time-Zone Offsets

```
data _null_;
   x=time();
   put x=e8601lz.;
run;
```

In This Example

- Because time values are scalar, SAS does not normally compute time values based on the time zone of the programmer's location. One exception to this rule is when a SAS time (not a datetime) is computed and then formatted with either the B8601LZ*w*. format or the E8601LZ*w*. format, as shown in the example above.
- These two formats query the SAS host code to determine the offset. Then the current local time and the offset (based on your time zone) display accordingly.

Note: If either B8601LZ*w*. or E8601LZ*w*. attempts to format a time outside of the time range 0 and 23:59:59, the time is formatted with asterisks to indicate that the value is out of range.

Output

The resulting value for X is **`11:41:54-04:00`**. **Note:** Your output value will be based on your time zone and the time at which you run your DATA step.

Example 7: Reading and Writing Time Values

You can read time values that do not have time-zone offset values into SAS time values using the B8601TM*w.d* and E8601TM*w.d* informats, as shown in this example:

```
data _null_;
   x=input('12:34:56',e8601tm8.);
   put x=b8601tm8. x=e8601tm10.;
run;
```

In This Example

- The B8601TM*w.d* and E8601TM*w.d* informats read the time values into SAS time values.
- The equivalent (like-named) formats write the time values to the SAS log for the variable X.

Output

The resulting values for X are as follows:

- **`123456`**
- **`12:34:56`**

DURATIONS

A *duration* is the period of time that is the difference between two time points. Durations can assume the same forms as the date, time, and datetime structures that are discussed previously.

In basic and extended notation, an uppercase P at the beginning signals that a duration follows.

Basic notation P*yyyymmdd*T*hhmmss* (can be positive or negative)

Extended notation P*yyyy-mm-dd*T*hh*:*mm*:*ss* (can be positive or negative)

A date value in *yyyy-mm-dd* form indicates a specific date in history. However, a duration value, similar to the following example, expresses a period of time.

`P0000-00-04` (indicates the span of zero years plus zero months plus four days)

Notice that all of the placeholders have a value, even if the value is zero.

The following example is the most common way to represent a basic and extended duration:

P*n*Y*n*M*n*DT*n*H*n*M*n*S

In this syntax, *n* is either 0 or a positive number, specifying the number of years (Y), months (M), days (D), hours (H), minutes (M), and seconds (S).

In addition, P*n*W represents duration as the number of weeks (W).

Pointers

- The W (weeks) in a duration can appear only when it is the sole component. For example, **`P1W2D`** is not permitted.
- Any of the *n* components can be omitted. For example, suppose you have the value **`P0Y0M3DT2H`**. You can omit the components that have a **`0`** value, as shown here:

  `P3DT2H` (indicates a duration of 3 days and 2 hours)
- If the time is unknown, it is permissible to omit it. In that case, the T must also be omitted, as shown in this example:

  `P3D` (indicates a span of 3 days)

(list continued)

- The T time delimiter is required if a time is specified because M refers to months in the date portion and it refers to minutes in the time portion.
- The lowest-order components (*n*) can be represented as fractions. For example, **P6.5W** specifies 6 ½ weeks.

INTERVALS

An *interval* comprises two values that represent the beginning and ending of an event, and it is a duration that is anchored to a specific point in time. Intervals are represented in the following forms:

- *datetime/datetime*
- *datetime/duration*
- *duration/datetime*

For example, an interval that is defined as "starting at 9:30am on April 2, 2012 for a duration of one hour" can be shown in either of the following ways:

- **2012-04-02T09:30:00/2012-04-02T10:30:00**
- **2012-04-02T09:30:00/PT1H**
- **PT1H/2012-04-020T10:30:00**

PARTIAL AND MISSING COMPONENTS

Clinical-trial data seeks to be as complete as possible, realizing that the precision of the data is based on the presence or absence of components in the date and time values. The year must always be four digits in length and a T precedes any time components. Complete values show all components with applicable values while hyphens delimit the date components and colons delimit time components. Here are some examples:

- **2012-03-25T22:14:16** (March 25, 2012 10:14:16 p.m.)
- **2012-03-25T22:15:16+03:00** (March 25, 2012 10:15:16 p.m. in the time zone GMT + 3 hours)
- **P2Y3M4DT7H8M9S** (A span of 2 years, 3 months, 4 days, 7 hours, 8 minutes, 9 seconds)

When any component of the date or time is not provided, it is called a *partial value*, and the components are considered missing. A missing component within the value should be represented with a hyphen (**-**) or an **x** so that it is easily readable and understood. A single hyphen represents the entire value for a given component. For example, one single hyphen can replace a four-digit year. If the time portion is omitted when a date value is specified, the T must also be omitted.

Durations can be expressed either as a span of time (as shown in the examples above) or in the long form as a datetime, but a mixture of the two forms within the same value is not allowed. Missing components should not be confused with zero values. The durations **P3D** and **P0000-00-03** are not the same because a component value of **0** is not the same as a missing component value. Change instances of **0** to **x** (**Pxxxx-xx-03**), and now this value is considered the equivalent of **P3D**.

Valid Value

P2Y4DT7H9S (other valid options for this value: **P0002-xx-04T07:xx:09** or **P0002---04T07:-:09**)

Invalid Value

P2Y---04T7H:-:9S

SAS can read truncated duration, datetime, and interval values, where one or more lower-order components are truncated because the value is **0** or the value is not significant. When you read in values that contain a time-zone offset, omitted components are not allowed. Therefore, you should use **00** in place of omitted components.

Examples of truncated values:

- **2012---18** (The 18th day of an unknown month in the year 2012. The time value is truncated.)
- **xxxx-xx-01T10** or **----01T10** (10:00 a.m. on the first day of each month. Minutes and seconds are truncated.)
- **2012-03** (The month of March 2012. The day and all time components are truncated.)

(list continued)

- **`--02--T-:23`** (The 23rd minute of unknown hour of unknown day of the second month of unknown year. Seconds are truncated.)
- **`2012-05-15T15:00:00+05:00`** (Because an offset is specified, hours and minutes cannot be omitted.)

THE $N8601B AND $N8601E INFORMATS AND FORMATS

Up to this point, this paper has discussed various SAS informats that are used to read values into SAS date, time, and datetime variables. The discussion also included the SAS formats that are used to write these values to a readable form. In addition to storing date and time values as numeric variables, SAS provides functionality for storing ISO 8601 durations, intervals, and datetime values as text strings to guarantee that all of the components are preserved, even when some are missing. The SAS informats *$N8601Bw.d* and *$N8601Ew.d* convert a duration, an interval, or a datetime value to what is referred to as an *entity*. The result is a binary value that is stored as a hexadecimal value that is not visually recognizable.

The $N8601B informat reads values in basic and extended notations, whereas the $N8601E informat reads values in the extended format only. Unlike the $N8601B informat, $N8601B reads single-digit components that do not supply leading zeros.

The following table illustrates notations and examples for various types of values that are read with $N8601B*w.d* and $N8601E*w.d* informats:

Value Type	ISO 8601 Notation	Example
Duration (Basic Notation)	P*YYYYMMDD*T*hhmmss*	**P20120513T123456**
Duration (Extended Notation)	P*YYYY-MM-DD*T*hh*:*mm*:*ss*	**P2012-05-13T12:34:56**
Duration	P*n*Y*n*M*n*DT*n*H*n*M*n*S	**P3y6m4dT12h34m56s**
Interval (Basic Notation)	*YYYYMMDD*T*hhmmss*/*YYYYMMDD*T*hhmmss* P*n*Y*n*M*n*DT*n*H*n*M*n*S/*YYYYMMDD*T*hhmmss* *YYYYMMDD*T*hhmmss*/P*n*Y*n*M*n*DT*n*H*n*M*n*S	**20120513T123456/20120613T112345** **P3y6m4dT12h34m56s/20120513T113423** **20120513T123456/P3y6m4dT11h23m45s**
Interval Extended Notation)	*YYYY-MM-DD*T*hh*:*mm*:*ss*/*YYYY-MM-DD*T*hh*:*m*:*ss* P*n*Y*n*M*n*DT*n*H*n*M*n*S/*YYYY-MM-DD*T*hh*:*mm*:*ss* *YYYY-MM-DD*T*hh*:*mm*:*ss*/P*n*Y*n*M*n*DT*n*H*n*M*n*S	**2012-05-13T12:34:56/2012-05-16T14:32:23** **P3y6m4dT11h23m45s/2012-08-11T14:22:00** **2012-07-04T12:33:22/P2y1m3dT6h2m1s**
Datetime (Basic Notation)	*YYYYMMDD*T*hhmmss.fff*+|-*hhmm*	**20120513T123456**
Datetime (Extended Notation)	*YYYY-MM-DD*T*hh*:*mm*:*ss.fff*+|_*hhmm*	**2012-05-13T12:34:56**

Table 3. Value Types That Can Be Read with the $N8601B*w.d* and $N8601E*w.d* Informats

After the $N8601B*w*.and $N8601E*w*. informats read the values into SAS, the equivalent (like-named) formats translate these entities into a meaningful display as datetime, duration, or interval values.

Informats That Read ISO 8601 Duration, Datetime, and Interval Values

- $N8601B*w.d* reads values in basic or extended format.
- $N8601E*w.d* reads values in extended format.

Formats That Write ISO 8601 Duration, Datetime, and Interval Forms

- $N8601B*w.d* writes the basic notations P*n*Y*n*M*n*DT*n*H*n*M*n*S and *yyyymmdd*T*hhmmss.*
- $N8601E*w.d* writes the extended notations P*n*Y*n*M*n*DT*n*H*n*M*n*S and *yyyy-mm-dd*T*hh*:*mm*:*ss*.

Other Valid Formats

- $N8601BA*w.d* writes P*yyyymmdd*T*hhmmss* and *yyyymmdd*T*hhmmss*.
- $N8601EA*w.d* writes P*yyyy-mm-dd*Thh:*mm*:*ss* and *yyyy-mm-dd*T*hh*:*mm*:*ss*.
- $N8601EH*w.d* writes same as $N8601EA*w.d* except uses a hyphen (-) for omitted components.
- $N8601EX*w.d* writes same as $N8601EA*w.d* except uses an x for each digit of an omitted component.
- $N8601H*w.d* writes same as $N8601E*w.d*, dropping omitted components in duration values and uses a hyphen (-) for omitted components in datetime values.
- $N8601X*w.d* writes same as $N8601E*w.d,* dropping omitted components in duration value and using an x for each digit of an omitted component in datetime values.

The following table compares some original values that are read into SAS with the $N8601B*w.d* informat and how the values are displayed with various formats.

Format That Is Applied to Original Values	Formatted Value (for the Original Value **2009-03-25**)	Formatted Value (for the Original Value **P1D**)	Formatted Value (for the Original Value **P0000-00-01**)
$N8601B	**20090325**	**P1D**	**P0Y0M1D**
$N8601E	**2009-03-25**	**P1D**	**P0Y0M1D**
$N8601BA	**20090325**	**P----01**	**P0000001**
$N8601EA	**2009-03-25**	**P----01**	**P0000-00-01**
$N8601EH	**2009-03-25T-:-:-**	**P----01T-:-:-**	**P0000-00-01T-:-:-**
$N8601EX	**2009-03-25Txx:xx:xx**	**Pxxxx-xx-01Txx:xx:xx**	**P0000-00-01Txx:xx:xx**
$N8601H	**2009-03-25T-:-:-**	**P1D**	**P0Y0M1D**
$N8601X	**2009-03-25Txx:xx:xx**	**P1D**	**P0Y0M1D**

Table 4. Output of Datetime and Duration Values with $N8601- Formats

THE CALL IS8601_CONVERT ROUTINE

After your values are stored as SAS variables, you can calculate intervals, durations, and datetimes with the CALL IS8601_CONVERT routine. The basic syntax for this routine is as follows:

CALL IS8601_CONVERT(*convert-from*, *convert-to*, <*from-variables*> , <*to-variables*>, <*date-time-replacements*>)

The arguments for this syntax are as follows:

Required Arguments[1]

convert-from accepts the following values:

'intvl' specifies the source value for the conversion is an interval value.

'dt/du' specifies the source value for the conversion is a datetime/duration value.

(list continued)

[1] SAS Institute Inc. *SAS® 9.3 Functions and CALL Routines: Reference*. (Cary, NC: SAS Institute Inc., 2011), Adapted from pages 164-165.

‘du/dt’ specifies the source value for the conversion is a duration/datetime value.

‘dt/dt’ specifies the source value for the conversion is a datetime/datetime value.

‘du’ specifies the source value for the conversion is a duration value.

convert-to accepts the following values:

‘intvl’ specifies to create an interval value.

‘dt/du’ specifies to create a datetime/duration interval.

‘du/dt’ specifies to create a duration/datetime interval.

‘dt/dt’ specifies to create a datetime/datetime interval.

‘du’ specifies to create a duration.

‘start’ specifies to create a value that is the beginning datetime or duration of an interval value.

‘end’ specifies to create a value that is the ending datetime or duration of an interval value.

Optional Arguments

from-variable specifies one or two variables that contain the source value.

to-variable specifies one or two variables that contain the converted values.

date-time-replacements specifies date or time components to use when a month, day, or time component is omitted from an interval, datetime, or duration value. The values are specified as a series of numbers, separated by a comma, in this order: year, month, day, hour, minute, or second.

The first argument to the CALL routine can be one or two values, and that number of values is based on how many variables you provide to the routine for an expected result. For example, datetime and duration values can be specified with an expected output of an interval. In this example, the supplied first argument would be `'dt/du'`. If two datetime values are supplied with an expected duration as output, the first argument would be `'dt/dt'`.

The second argument to the CALL routine can also be one or two values, depending on what type of result you expect SAS to compute using the input you supply in the first argument.

Although the routine is robust and converts intervals, durations, and datetime values, it is also a handy tool for performing calculations with dates and times. The following sections provide examples of calculations using the CALL IS8601_CONVERT routine.

Note: When you create durations, the minimum length is 16 whereas the minimum length for intervals is 32.

Example 1: Converting a Duration to a SAS Time

Suppose you supply the duration value **P8W**, which specifies eight weeks. Such a value is often coupled with a datetime value to produce a duration, but it can be used alone and converted to a SAS time. There is no SAS informat to convert this **P8W** value to a SAS time, but you can use the CALL IS8601_CONVERT routine instead.

Consider the following DATA step:

```
data a;
   x='P8w';
   call is8601_convert('du','du',x,mynew);
   put mynew=time8.;
run;
```

In This Example

- The X variable is a duration. Therefore, `'du'` is the first argument.
- A time value is expected as output, but there is no *'convert-to'* value for time. As a result, the value `du` is used for duration. Notice that both arguments require single quotation marks.
- X is the variable name that is supplied, and MYNEW is the variable being created.
- Because the result is a SAS time value, you can use a SAS time format (TIME8.).

Output

The resulting value for MYNEW is **`1344:00`**, which indicates 1344 hours.

Example 2: Converting a Duration and a Datetime to an Interval

This example takes the previous example one step further, using the same duration value with a datetime value to output an interval. The following DATA step is based on an event that lasts for eight weeks, ending on February 11, 2012 at 12:22 pm.

```
data a;
   length mynew $32;
   x='P8w';
   y='11feb2012:12:22'dt;
   call is8601_convert('du/dt','intvl',x,y,mynew);
   put mynew=$n8601e.;
run;
```

In This Example

- The first argument (`'du/dt'`) to the CALL IS8601_CONVERT routine indicates the types of variables that are being passed in for conversion. The value `'du/dt'` specifies that two variables are being passed: one is a duration (X) value and the other is a datetime (Y) value. Because the duration value comes before the datetime value, the datetime value is assumed to be the end of the interval.
- The result that you want from this DATA step is an interval. Therefore, the second argument is `'intvl'`.
- The remaining arguments name the incoming variables (X and Y) and the new variable (MYNEW) that is being created. The order of these variables must match the types of variables that are specified in the first argument. For example, if X and Y are reversed, the following note will appear in the log:

```
NOTE: Invalid argument to function IS8601_CONVERT at line 1661 column 6.
mynew=************************************************
mynew=  x=P8w y=1676204520 _ERROR_=1 _N_=1]
```

Output

The resulting value for MYNEW is **`P8W/2012-02-11T12:22:00.000`**.

Example 3: Computing the Start Datetime of an Interval When You Have a Duration and a Datetime

Example 2 above computes an interval when datetime and duration values are supplied. The next example computes the starting datetime for an interval when a duration and datetime are supplied.

```
data a;
   length mynew $16;
   x='P8w';
   y='11feb2012:12:22'dt;
   call is8601_convert('du/dt','start',x,y,mynew);
      /* If the LENGTH statement above is commented out, replace the PUT */
      /* statement below with put mynew datetime22.;                      */
   put mynew=$n8601e.;
run;
```

In This Example

- The CALL IS8601_CONVERT routine uses the keyword `start` as the second argument in order to compute the starting datetime value for the interval.
- Because the SAS datetime value is computed, you can remove the LENGTH statement so that MYNEW is created as a numeric variable. If you remove that statement, you need to add a PUT statement that specifies the DATETIME22. format. It is also valid to leave the LENGTH statement and output the variable using the $N8601E*w*. format, as shown in the second PUT statement.

Output

The resulting value for MYNEW is **2011-12-17T00:00:00.000**.

Example 4: Converting a Duration of One Type to a Duration of a Different Type

If you receive data that is represented in hours, you can perform a conversion similar to the previous example to obtain output as a duration.

```
data _null_;
   x='1271:59:00';
   time=input(x,time10.);
   length dur $16;
   call is8601_convert('du','du',time,dur);
   put dur=$n8601e.;
run;
```

In This Example

- The TIME*w.d* informat reads the time value into SAS.
- Then the CALL IS8601_CONVERT routine converts the time value to a duration value.

Output

The resulting duration value is **P0Y1M22DT23H59M0.0S**.

Example 5: Converting Two Datetime Values to a Duration

This next example determines the amount of time between two datetime values, and it outputs a duration.

```
data a;
   length dur $16;
   start='02apr2012:12:30:22'dt;
   end='08apr2012:14:32:22'dt;
   call is8601_convert('dt/dt','du',start,end,dur);
   put dur=$n8601e.;
run;
```

In This Example

- The CALL IS8601_CONVERT routine uses the following arguments:
  - `'dt/dt'` indicates that two datetime values are being passed into the function.
  - `'du'` indicates that a duration value is the expected output.
  - `'start'` is the name of the first datetime variable.
  - `'end'` is the name of the second datetime variable.
  - `'dur'` is the name of the desired output variable.
- A LENGTH statement is required in order to specify that DUR is a character variable and that the length should be 16. If you create an interval, a length of 32 is required.
- The routine creates an undecipherable hexadecimal value. Therefore, it is necessary to format the value with one of the $N8601 formats; in this case $N8601E*w.d*.

Output

The resulting value is **P6DT2H2M**.

Example 6: Converting an Interval to a Datetime and a Duration

You can specify an interval value as two datetime values separated by a forward slash (/), which separates the beginning and ending values for an event, as shown in this example:

```
data _null_;
   length final2 $16;
   int=input('2012-03-15T14:32:00/2012-03-29T09:45:00',$n8601e40.);
   call is8601_convert('intvl','dt/du',int,final1,final2);
   put final1= final2=$n8601e.;
run;
```

In This Example

- The datetime values are specified inside single quotation marks so that the INPUT function and the $N8601E*w*. informat can convert the values into an interval value in the variable INT.
- CALL IS8601_CONVERT converts that interval value into two variables: FINAL1 is a datetime variable and FINAL2 is a duration variable.
- Because a SAS datetime variable (FINAL1) is a numeric variable that can be stored in 8 bytes, a length value is not required for this variable in the LENGTH statement. However, a duration requires a length of 16; therefore, a length is specified for FINAL2 in the LENGTH statement.
- Because FINAL2 is a duration, the $N8601Ew. format is used to write its value to the log so that the output is understandable.

Output

The resulting values for FINAL1 and FINAL2 are as follows:

- FINAL1: **1647441120**#
- FINAL2:#**P13DT19H13M**#

Pointer

If the first datetime value in the example had been missing the month value (**2012---15T14:32:00**), you could supply that value by specifying it in the last parameters of the CALL routine, as shown here:

```
call is8601_convert('intvl','dt/du',int,final1,final2,,2);
```

Notice that the month is specified here as **2**, and the two consecutive commas preceding that value indicate that a year value was not supplied. Therefore, a datetime and a duration will be computed based on the date **2012-02-15T14:32:00** and the other date specified in the code. In this case, the output from the PUT statement for each variable is as follows:

- FINAL1: **1644935520**#
- FINAL2:#**P1M13DT19H13M**

Example 7: Handling Missing Components

It is not uncommon for one or more date or time components to have missing values. Yet you still need to compute the output. The following example illustrates how to handle such a situation:

```
options missing='-';
data temp;
   input y mo d h min s;
   length text $19;
   text=cats(y,'-',mo,'-',d,'T',h,':',min,':',s);
   entity=input(text,$n8601b19.);
   put entity=$n8601e.;
   datalines;
2011 6 . 10 15 20
2011 . 5 10 15 20
2011 6 30 10 15 20
;
run;
```

In This Example

- Because missing components should be indicated with a hyphen (**-**) or the letter **x**, the OPTIONS statement specifies the MISSING= system option so that all missing numeric values are output as a hyphen. This works equally if the option is set to **x**.
- The CATS function concatenates all of the components, adding the hyphen and colon delimiters between the date and time values to form the value for the TEXT variable.
- The TEXT value is read by the $N8601B*w*. informat to create the ENTITY variable.
- Because this created ENTITY value is a hexadecimal string, it is formatted with the $N8601E*w*. format to produce readable values that show a hyphen where missing values exist.

Output

The resulting values for ENTITY are as follows:

- **2011-06--T10:15:20**
- **2011---05T10:15:20**
- **2011-06-30T10:15:20**

Pointer

In this example if ENTITY had been created with the $N8601E19. informat, the log would show notes indicating that there are invalid arguments in the INPUT function as a result of single-digit month and day values. To eliminate this problem, you can use the $N8601B*w*. informat to read the single-digit values correctly.

Changes are being considered for a later SAS release to enable the CALL IS8601_CONVERT routine to handle missing components so that a workaround such as the one described above is not required.

CONCLUSION

The ISO 8601 standards are visible throughout all clinical trials when you are representing dates, times, durations, and intervals. The SAS suite of informats and formats, along with the CALL IS8601_CONVERT routine, enable you to reference and use these entities in a variety of ways. These features allow various forms of values to be read and stored as SAS variables, to be manipulated in SAS, and then to be output in a variety of ways. The functionality gained from the multifaceted CALL routine means direct creation of various types of values that otherwise take multiple lines of code to accomplish.

ACKNOWLEDGMENTS

I would like to thank Frank Roediger, Gene Lightfoot, and Russ Lavery for sharing their wealth of knowledge about clinical trials. Rick Langston, the SAS developer, has been a tremendous resource since this functionality became available.

REFERENCE

SAS Institute Inc. *SAS® 9.3 Functions and CALL Routines: Reference*. (Cary, NC: SAS Institute Inc., 2011), 164-165. Available at **support.sas.com/documentation/cdl/en/lefunctionsref/63354/PDF/default/lefunctionsref.pdf**.

RECOMMENDED READING

SAS Institute Inc. 2011. *SAS® 9.3 Formats and Informats: Reference.* Available at **support.sas.com/documentation/cdl/en/leforinforref/63324/HTML/default/viewer.htm#titlepage.htm.** Accessed on March 27, 2012.

SAS Institute Inc. 2011. *SAS® 9.3 Functions and CALL Routines: Reference*. (Cary, NC: SAS Institute Inc., 2011). Available at **support.sas.com/documentation/cdl/en/lefunctionsref/63354/PDF/default/lefunctionsref.pdf**

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Kim Wilson
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
E-mail: support@sas.com
Web: support.sas.com