PharmaSUG 2026 - Paper DS-174

ISO 8601 and SAS®—and R! A Practical Approach

Derek Morgan, Bristol Myers Squibb
Marckenley Mercie, Bristol Myers Squibb

ABSTRACT

The ISO 8601 standard for dates and times has long been adopted for clinical data by regulatory agencies around the world. While there are many homemade solutions for working in this standard, SAS has many built-in solutions, from formats and informats to the IS8601_CONVERT routine, which painlessly handles durations and intervals. As R gains popularity, several packages are available to handle this mandated standard. This paper will detail both SAS and R capabilities around this standard.

INTRODUCTION

This paper will help you understand the background and rationale for the ISO 8601 standard. Then we'll provide some methods and lesser-known shortcuts to facilitate using SAS and R with this standard. In this paper, the word "datetime" will be used to refer to a SAS datetime value (seconds since January 1, 1960). The hyphenated "date-time" will be used when referring to the specification of combined date and time according to the ISO 8601 standard. When we discuss R, we will explore how R natively represents ISO 8601 date-time values and highlight *lubridate* as a widely adopted package for working with dates and times.

WHAT IS ISO 8601?

It is an internationally accepted methodology that only uses numbers to describe dates and times, and to facilitate the exchange of data, particularly between international parties. The standard removes the issue of translating month names (23 lip 2021) and defines the order of date and time elements to remove confusion due to local norms. For example, the string "03-08-21" can be translated as 3 possible dates: March 8, 2021 (assuming the date is in the 21st century); August 21, 2003, or even August 3, 2021, and it depends on the cultural norms of the date's location. The standard has two forms, basic and extended. The basic form provides the numbers without delimiters, while the extended form uses delimiters such as dashes, colons, and periods. Table 1 illustrates the difference:

Basic Notation		Extended Notation
2016		2016
201606		2016-06
20160614		2016-06-14
20160614T1422		2016-06-14T14:22
20160614T142237		2016-06-14T14:22:37
20160614T14223774		2016-06-14T14:22:37.74

Table 1: ISO 8601 Basic vs. Extended Notation

The complete standard (available from http://www.iso.org/iso/home.html) covers the following:

- Date
- Time of day
- Date and time
- Coordinated universal time (UTC)
- Local time with offset to UTC
- Time intervals
- Recurring time intervals

This standard has been adopted by the Clinical Data Interchange Standards Consortium (CDISC), which makes it relevant to the work many of us do every day. However, it should be emphasized that the standard is a global one with multiple applications. Therefore, it is not restricted to clinical data and the CDISC realm. This paper focuses on the discussion of examples clinical programmers are likely to encounter, and will use the extended ISO notation throughout. It is much easier to understand the delimited form when viewing ISO dates and times. Thankfully, the CDISC standard mandates the use of the extended form.

ISO 8601 DATE, TIME OF DAY, AND DATE-TIME

When it comes to dates and times, what are the most important features of this standard? First, and most important, it supports partial information. The standard accepts missing components in dates and times. This makes sense; you generally don't get the seconds component when you ask about time, and if someone asked you about the vacation you took in 2015, you can probably give them the month without looking, but not the exact date and time you left the house. This is probably the largest issue when working with the standard in a computing environment, since most date algorithms only work when given specific dates and/or times. In most cases, you won't lose accuracy if you use zero for missing seconds, but that is either an oversimplification or impossible for other missing components.

Second, ISO date, time, and date-time values sort correctly without transforming into a numeric variable (try it!) I use ISO dates in file names if I need to keep multiple versions of a file and want to easily see the date of the file, regardless of environment. I can sort by file name, and they will show in chronological order. One warning: you should be aware that the sorting logic around missing components could depend on your computing environment (or the SORT algorithm options in effect.)

ISO 8601 AND LOCAL TIME (WHAT IS UTC?)

Since time of day is relative to geographic location, the standard allows you to express local time as a value with an offset from a common reference point. This reference point is referred to as "Coordinated Universal Time", or "UTC". UTC is the name of the technical standard, and it is currently set to the value of Greenwich Mean Time (GMT). In present practice, both "UTC" and "GMT" are equivalent and are used interchangeably. The major caveat with using offsets is not every offset is equivalent to a whole hour, so if you are using offsets, make sure you display hours and minutes. Remember, "a.m." and "p.m." are not part of the standard, so time will always be expressed in a 24-hour clock.

ISO 8601 TIME INTERVALS

It is important to understand the term "interval" does not mean the same thing between SAS and the ISO standard. A SAS interval defines a commonly referenced aggregate period such as "year" or "quarter." Within the standard, "interval" defines a connection between two specific dates, times, or date-times.

There are several ways to represent durations and intervals in line with the standard. Any of these is equally valid, but the one we see most often in our line of work is the ISO duration form **P**n**Y**n**M**n**DT**n**H**n**M**n**S**. If the duration is less than one day, this is shown as **PT**n**H**n**M**n**S**. If there is no time value, then it will be **P**n**Y**n**M**n**D**. The final duration form is a special case and only used when the duration is expressed in weeks: **P**n**W**. As number of weeks is the only component in this special case, there are no delimiters used, so it is the same in both basic and extended forms.

The letter **P** stands for "period", and **T** is the ISO 8601 delimiter signifying a time value follows. Leading or trailing components whose values are zero are optional (e.g., "P4M6D" for 4 months, 6 days is acceptable, but "P1Y0M6D" for one year and 6 days.) If the ending date-time is prior to the starting date-time, the P will be prefaced with a minus (-) sign.

The **PT**n**H**n**M**n**S** form can cause confusion in those unfamiliar with the standard; the "P" has been mistaken for the analysis period, or when followed by T, it has been interpreted as "partial time." Neither interpretation is correct.

Examples 1-4 below use the same starting and ending date-time values and extended notation for each corresponding row, so you may replicate the results if desired. Example 1 shows the ISO duration form:

	AE Start Date-time (in ISO 8601 format)	AE End Date-time (in ISO 8601 format)	ISO 8601 Duration Form
1	2020-08-05T17:19:00	2020-08-06T11:00:00	PT17H41M
2	2020-08-11T22:30:00	2020-08-14T09:00:00	P2DT10H30M
3	2020-08-05T09:25:00	2020-08-11T22:15:00	P6DT12H50M
4	2020-08-23T07:00:00	2020-08-26T07:58:00	P3DT58M
5	2020-07-29T09:34:00	2020-07-30T07:30:00	PT21H56M

Example 1: ISO Duration Form

ISO 8601 intervals (again, not to be confused with SAS intervals,) can be represented in in three ways: the starting and ending ISO date-times of the interval, or the starting ISO datetime and an ISO duration, or an ISO duration followed by the ending ISO datetime. A slash (/) is the delimiter between the elements. Example 2 shows the date-time/datetime form:

	AE Start Date-time (in ISO 8601 format)	AE End Date-time (in ISO 8601 format)	ISO 8601 Interval (date-time/date-time form)
1	2020-08-05T17:19:00	2020-08-06T11:00:00	2020-08-05T17:19/2020-08-06T11:00
2	2020-08-11T22:30:00	2020-08-14T09:00:00	2020-08-11T22:30/2020-08-14T09:00
3	2020-08-05T09:25:00	2020-08-11T22:15:00	2020-08-05T09:25/2020-08-11T22:15
4	2020-08-23T07:00:00	2020-08-26T07:58:00	2020-08-23T07:00/2020-08-26T07:58
5	2020-07-29T09:34:00	2020-07-30T07:30:00	2020-07-29T09:34/2020-07-30T07:30

Example 2: Start/End Date-time Interval Form

Example 3 and Example 4 show the date-time/duration ISO interval forms, delimited by slashes (/):

	AE Start Datetime (in ISO 8601 format)	AE End Datetime (in ISO 8601 format)	ISO 8601 Interval (date-time/duration form)
1	2020-08-05T17:19:00	2020-08-06T11:00:00	2020-08-05T17:19/P0Y0M0DT17H41M0S
2	2020-08-11T22:30:00	2020-08-14T09:00:00	2020-08-11T22:30/P0Y0M2DT10H30M0S
3	2020-08-05T09:25:00	2020-08-11T22:15:00	2020-08-05T09:25/P0Y0M6DT12H50M0S
4	2020-08-23T07:00:00	2020-08-26T07:58:00	2020-08-23T07:00/P0Y0M3DT0H58M0S
5	2020-07-29T09:34:00	2020-07-30T07:30:00	2020-07-29T09:34/P0Y0M0DT21H56M0S

Example 3: Date-time/Duration Interval Form

AE Start Datetime (in ISO 8601 format)	AE End Datetime (in ISO 8601 format)	ISO8601 Interval (duration/date-time form)
2020-08-05T17:19:00	2020-08-06T11:00:00	P0Y0M0DT17H41M0S/2020-08-06T11:00
2020-08-11T22:30:00	2020-08-14T09:00:00	P0Y0M2DT10H30M0S/2020-08-14T09:00
2020-08-05T09:25:00	2020-08-11T22:15:00	P0Y0M6DT12H50M0S/2020-08-11T22:15
2020-08-23T07:00:00	2020-08-26T07:58:00	P0Y0M3DT0H58M0S/2020-08-26T07:58
2020-07-29T09:34:00	2020-07-30T07:30:00	P0Y0M0DT21H56M0S/2020-07-30T07:30

Example 4: ISO Duration/Date-time Interval Form

There is more to the standard than the brief examples above, but these are ones most relevant to this paper. Next, we will discuss how to work with the standard in SAS.

SAS AND ISO 8601

SAS has built-in formats and informats to handle a variety of date and time displays. It fully supports the ISO 8601 standard. SAS will reliably display its date, time, and datetime values to the standard's specifications in both basic and extended forms. Conversely, if the ISO value does not have any missing components, SAS will translate ISO 8601 dates, times and date-time values into SAS date values. SAS dates, times, and datetimes do not allow for missing components because its date algorithm stores values as days since January 1, 1960, seconds since midnight, or seconds since midnight, January 1, 1960.

WHAT DO WE DO ABOUT THAT?

Most of the time we use imputation to create values for missing components in ISO 8601 data when we need to have a SAS value. This allows you to start with a simple premise: read the ISO date, time, or date-time string into a SAS variable with an informat. If the resulting SAS value is missing, then you should perform imputation as specified by the analysis plan. This is the main reason I prefer not to create a "do-everything" ISO 8601 date processing macro. Such a macro would need at least one parameter representing the desired imputation algorithm and type of value being imputed, such as year, month, and/or day. This could become a nightmare. An unanticipated imputation method might not cause an error, but it would most likely deliver an incorrect result. If you checked the validity of the imputation method at the start of the macro and errored out on an unknown method before it executed, then you have a macro that won't work, and must be sent back to the developer while your study clock keeps ticking. Richann Watson has developed a SAS function using PROC FCMP to perform the processing of dates with imputations for missing values.

READING ISO 8601 DATES AND DATETIMES

Two SAS informats will handle the creation of SAS date and datetime variables as long as you have an ISO 8601 date-time string. The E8601DT. informat will create a SAS datetime from the ISO string, and the E8601DA. informat will create SAS dates—without parsing the ISO date-time string. Most of my programs use some variation of the code in Sample Code 1 to create SAS variables from ISO 8601 strings (shown in DATA step syntax, but the INPUT statement is the same in SQL):

```
1. xxxdtm = INPUT(xxDTC,E8601DT.); /* Datetime */
2. xxxdt = INPUT(xxDTC,E8601DA.);  /* Date */
3. xxxtm = TIMEPART(xxxdtm);       /* Time */
```

Sample Code 1: Creating SAS Variables from an ISO Date-time String

Line 1 creates a SAS datetime variable from the ISO string. If the ISO string is not a complete date-time string, then the result will be missing. This makes sense. Line 2 creates a SAS date variable from the ISO string. It will be missing if any date component is missing from the date-time string.

Since the ISO 8601 datetime and date informats read the strings without parsing, why not use one of the ISO 8601 time informats? Unfortunately, the ISO 8601 time informats are specifically designed to translate ISO time strings with or without time zones or offsets. Unlike the *8601DA. informat, there is no SAS informat that will extract the time from an ISO date-time string. However, we can still avoid parsing by using the TIMEPART() function to extract the time from a SAS datetime value, which we conveniently created from the original ISO string in line 1. If the original ISO string does not have a time component, then it's missing, and you can proceed accordingly.

From here, you can decide if date or time imputation is necessary by a simple test. If the SAS datetime, date, or time is missing, then you need to run your imputation process. Three lines of code give you SAS datetime, date, and time values from an ISO 8601 datetime string, and let you know if imputation is necessary. Any errors arising during the processing of the original ISO string should be considered source data errors and should be inspected to determine the proper course of action.

This method assumes your ISO date-time strings are in extended notation. You will get an error if you try to read date or datetime values without delimiters ("basic" notation) with the extended notation informats. However, the above method does not work reliably with data in basic notation, and although the basic

notation informats will process extended notation data, you should not substitute them for the extended notation informats. The basic notation informats impute a missing month or day to 1, and will set a missing time to midnight instead of missing. The following examples demonstrate what happens when you try to process basic and extended ISO date-time strings with the two informats. Example 5 corresponds to line 1 in Sample Code 1:

```
DATA test_datetime;
INFILE datalines;
INPUT @1 extended E8601DT. @20 basic B8601DT.;
FORMAT extended basic DATETIME19.;
DATALINES;
2021-03-26T16:14   20210326T1614
2021-06-24         20210624
2021-09            202109
;
RUN;
```

And the result:

Obs	extended	basic
1	26MAR2021:16:14:00	26MAR2021:16:14:00
2		24JUN2021:00:00:00
3		01SEP2021:00:00:00

Example 5: Automatic Data Substitution in SAS Datetime Values with Basic Notation Informats

Here, the extended notation informat only reads the complete string with delimiters (observation 1), while the basic notation informat reads it without delimiters. However, the basic notation automatically imputes a datetime for the partial basic datetime strings in observations 2 and 3. This imputation always substitutes the value 1 for missing date components and zero for the missing time component, which may not be what you want.

Now, let's run the same ISO strings with the *8601DA. informats to create SAS date values. Example 6 corresponds to line 2 of Sample Code 1:

```
DATA test_date;
INFILE datalines;
INPUT @1 extended E8601DA. @20 basic B8601DA.;
FORMAT extended basic DATE9.;
DATALINES;
2021-03-26T16:14   20210326T1614
2021-06-24         20210624
2021-09            202109
;
RUN;
```

And the result:

Obs	extended	basic
1	26MAR2021	26MAR2021
2	24JUN2021	24JUN2021
3		01SEP2021

Example 6: Automatic Data Substitution in SAS Date Values with Basic Notation Informats

As you can see, both informats extracted the date from the ISO string. This time, the extended notation informat extracted the date from both the datetime and date string while leaving the partial date string

missing (with an attendant note in the SAS log.) The basic notation informat once again read all three, substituting the value 1 for the missing day in the last string. Again, this may not be what you want.

WRITING ISO 8601 DATES AND DATETIMES

If you're not using a format, you're doing this the hard way. Example 7 uses the example datetime of March 26, 2021, 4:14 P.M. from the previous section:

```
DATA write_datetime;
dtm_value = '26MAR2021:16:14'DT;
vsdtc = PUT(dtm_value,E8601DT.);
vsdt = PUT(dtm_value,E8601DN.);
vstm = PUT(TIMEPART(dtn_value),E8601TM.);
date_value = '15MAY2021'D;
vsdt2 = PUT(date_value,E8601DA.);
RUN;
```

The result:

dtm_value	vsdtc	vsdtm	vstm	date_value	vsdt2
1932394440	2021-03-26T16:14:00	2021-03-26	16:14:00	22415	2021-05-15

Example 7: Writing ISO 8601 Strings from SAS Values

It really is that simple. What if you don't want seconds in your datetime string? On the one hand, if you haven't collected them, they are technically "missing". On the other, it is generally assumed that filling this with zero seconds does not harm the integrity of the data, and it does allow you to create a complete SAS datetime value. The time value is less of an issue, because the TIME5. format is ISO 8601-compliant. What if you are really a stickler? Lop off the seconds with:

```
vsdtc = SUBSTR(PUT(dt_value,E8601DT16.),1,16);
```

Why do you need to use the SUBSTR() function? Why can't you just do:

```
vsdtc = SUBSTR(PUT(dt_value,E8601DT16.);
```

When originally developed, the E8601DT. format did not allow a length of less than 19. This has changed in a recent maintenance release of SAS, and now the format supports a minimum length of 16. However, as with all formats and informats, you should check the most current version of the SAS documentation. New formats and informats are added from time to time, and as has happened with this format, specifications surrounding existing ones may change.

PARTIAL DATES AND TIMES

We all know SAS stores dates and times in numbers. However, it is assumed that you have an exact date or time; otherwise, SAS cannot store a single value. Clinical data may not be so exact when it comes to dates and times, and you may not be able to impute a day, month, or even year for a missing component. Without imputation, your SAS date or time value will be missing. What do you do?

First, let me reiterate that SAS date, time, and datetime values are always stored as numeric values. There are NO exceptions. So how does SAS handle a standard for dates and times that explicitly accounts for missing date or time components?

You can store ISO data natively using character ISO informats to store the data in character values. However, your data isn't stored as the basic or extended notation character strings we are accustomed to. SAS stores them in its own internal representation. What does that look like? Example 8 will show you:

```
1.   DATA iso_internal;
2.   INFILE datalines PAD MISSOVER;
3.   INPUT raw_iso $ 1-20 @1 extended E8601DT.
           @1 iso8601_internal $N8601E.;
4.   extended_fmt = extended;
5.   iso8601_fmt = iso8601_internal;
6.   FORMAT extended_fmt DATETIME19. iso8601_fmt $N8601EA.;
7.   DATALINES;
8.   2019-03-20T15:05:30
9.   2020-12-22T06:40
10.  2018-01-15
11.  2021-09
12.  2021-02-15T02
13.  2020
14.  ;
15.  RUN;
```

	Raw ISO String	SAS Datetime Value	SAS ISO 8601 Internal Value	Formatted SAS Datetime Value	Formatted SAS ISO 8601 Internal Value
A	2019-03-20T15:05:30	1868713530	2019320150530FFD	20MAR2019:15:05:30	2019-03-20T15:05:30
B	2020-12-22T06:40	1924238400	2020C220640FFFFD	22DEC2020:06:40:00	2020-12-22T06:40
C	2018-01-15		2018115FFFFFFFFD		2018-01-15
D	2021-09		20219FFFFFFFFFFD		2021-09
E	2021-02-15T02		202121502FFFFFFD		2021-02-15T02
F	2020		2020FFFFFFFFFFFD		2020

Example 8: How Character ISO Informats Work

First, let's look at the code: The most important pieces are the formats and informats used in lines 3 and 6. We've already discussed using the E8601DT. informat to turn an ISO date-time string into a SAS datetime, but what does $N8601E. do? The "$" indicates this is a character informat, and the "8601E" says it's for the extended form of ISO 8601. This informat will process an ISO 8601 date-time, duration, or interval string and store it in SAS as ISO *data*, not just a string in a character variable. Similarly, the $N8601EA. format will decode this ISO data into something we understand. Character variables containing ISO data must be a minimum of 32 characters long.

The table in Example 8 shows how SAS stores ISO date-time values internally. The "Raw ISO String" column is a sample of what we might see in a --DTC variable. As you can see, lines A and B are the only complete ISO 8601 date-time strings. Therefore, they are the only rows that can be stored as SAS datetime values without imputation. Taken as they are, lines C through F do not have enough information to pin down an exact time on an exact day, so the SAS datetime value is missing. The "SAS ISO 8601 Internal Value" column displays the ISO data values SAS stored in the variable ISO8601_INTERNAL. Finally, the last column shows you get the original --DTC string when you format the stored ISO 8601 value.

Since the first column and the last column are the same, why would you want to waste the processing to store a --DTC string in an internal representation that needs to be formatted to make any sense? You must remember that ISO 8601 is an all-encompassing international standard, not merely a CDISC one. The CDISC standard only uses ISO strings, so we don't need to store them as ISO data. Nonetheless, SAS has developed a routine to work with ISO data. This little-known routine has a wide range of capabilities.

WELCOME TO THE IS8601_CONVERT ROUTINE

This routine can manipulate date, time, ISO date-time, ISO interval, and ISO duration data. In short, it can handle every type of data specified in the standard, as well as SAS date, time, and datetime values. You can use the routine to create a SAS ISO 8601 value from individual date and time components (up to 6;

year, month, day, hours, minutes, seconds.) The complete (but general) syntax for the IS8601_CONVERT routine is:

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

One of the more convenient aspects of this routine is the ability to include substitutions for missing components inside the routine itself. You can specify these replacements by individual component, that is, you must specify a value to use for each of a missing year, month, day, hour, minute, or second. The default replacement value for year is missing. SAS will substitute 1 for both month and day by default, while 0 is the default substitution for hour, minute and second.

Replacements are separated by commas in the order *year, month, day, hour, minute, second*s, and you must provide a value or a comma as a placeholder if you are not going to change the default replacement for a date-time component. While this may be convenient, simple replacement may not be appropriate for your analysis purposes. Additionally, the same replacements are used for all input variables. You cannot have different replacements for each of the *from-variables*. There is no way to skip replacement. The routine will always use the default replacement values for missing date and/or time components.

If ISO strings are sufficient for CDISC, why should you make the effort to know about this routine? Let's look at ISO durations, most often seen in the --DUR variables from SDTM or added as an analysis variable in ADaM.

This little trick simplifies calculating and creating an ISO 8601 duration string beginning with the letter "P". Example 9 details the simple case where we have a complete datetime value. For this example, time is set to midnight of the given day. My recommendation would be to use this technique after you have performed any necessary imputation, so you don't have any partial datetime values, or at the least, partial dates, assuming that midnight is an acceptable replacement for missing time. Note that the variables AESTDTM and AEENDTM are SAS datetime values, and the duration variables AEDUR_UNF and AEDUR are character variables. Any result variable in this call must be character for the routine to work properly, otherwise, you'll get the result in seconds just as if you ran the statement: **aedur_= aendtm-astdtm;.**

```
1. DATA duration1;
2. SET full_datetimes;
3. LENGTH aedur aedur_unf $ 32; /* Important! If the result variable is
                                   numeric, AEDUR will be in seconds, i.e.,
                                   aeendtm – aestdtm */
4. CALL IS8601_CONVERT('dt/dt','du',astdtm,aendtm,aedur);
                     /* OR */
4. CALL IS8601_CONVERT('dt/dt','du',aestdtc,aeendtc,aedur);

5. aedur_unf = aedur;
6. FORMAT aestdtm aeendtm datetime19. aedur $N8601E.;
7. RUN;
```

The result below shows the event durations calculated above.

	Start Date/Time of Adverse Event	End Date/Time of Adverse Event	Duration of Adverse Event	Unformatted Duration/of Adverse Event (SAS ISO data)
A	20MAR2018:00:00:00	20MAR2018:00:00:00	P0W	000000000000000E
B	22DEC2019:00:00:00	23DEC2019:00:00:00	P1D	FFFFF01FFFFFFFFC
C	15JAN2020:00:00:00	04JUN2020:00:00:00	P4M20D	FFFF420FFFFFFFFC
D	25FEB2018:00:00:00	16MAY2020:00:00:00	P2Y2M21D	0002221FFFFFFFFC

Example 9: Using CALL IS8601_CONVERT to Create CDISC Duration Variables

Well, that was easy. This is how the parameters work in the call itself (line 4.) the first parameter, 'dt/dt' means both variables to be converted are datetime (or date-time) values, and the second parameter, 'du', tells the routine we want to calculate the result as an ISO duration value. The event start and end variables (in that order, otherwise you'll get negative durations) are followed by the result variable.

A few things to note here: first, the start and end date/time columns are right justified. They are numeric values (seconds since midnight, January 1, 1960,) and by default, the DATETIME. format is right justified. Next, row A shows the duration in weeks for the impossible case of an event that ended exactly when it started. You should never see an event of zero duration. Remember, these are ISO 8601 durations, not analysis durations. We generally add 1 day to the duration for analysis data in many situations to give events a minimum duration of one day. If you want to represent that analysis duration as an ISO 8601 duration, adjust the end date or datetime as appropriate.

What if you have partial dates? I strongly recommend performing any imputation as specified in the statistical analysis plan to create full datetime values before using the above method. However, you can take advantage of the automatic substitution for missing ISO date-time components in the CALL IS8601_CONVERT routine. You may use the default imputations or choose the values to be substituted for missing components. Example 10 creates ISO 8601 date-time data in SAS by using the $N8601E. informat. This example differs from Example 9 because of the incomplete dates. If you try to read them as SAS datetime values, you will get missing values and missing durations as a result. You can only read partial dates as ISO 8601 data, and SAS will do the substitution before calculating the duration result. This time, also note that in line 3, AESTDT, AEENDT and AEDUR are defined as *character* variables, because they represent ISO *data*.

```
1.   DATA duration2;
2.   INFILE datalines PAD MISSOVER;
3.   LENGTH aestdtc aestdtc_unf aeendt aeendt_unf aedur aedur_unf $ 32;
4.   INPUT @1 aestdtc :$N8601E. @12 aeendtc :$N8601E.;
5.   CALL IS8601_CONVERT('dt/dt','du',aestdtc,aeendtc,aedur);
6.   aestdt_unf = aestdt;
7.   aeendt_unf = aeendt;
8.   aedur_unf = aedur;
9.   FORMAT aestdt aeendt aedur $N8601E.;
10.  DATALINES;
11.  2018-03-20
12.  2020-12-22 2020-12-23
13.  2019-01-12 2019-06
14.  2018       2021-01
15.  2015-05    2019
16.  ;
17.  RUN;
```

And the result:

	Adverse Event Start Date	Unformatted Start Date-Time of Adverse Event in ISO Data Format	Adverse Event End Date	Unformatted Adverse Event End Date-Time in ISO Data Format	Adverse Event Duration	Unformatted Duration of Adverse Event
A	2018-03-20	2018320FFFFFFFFD	**********			FFFFFFFFFFFFFFF
B	2020-12-22	2020C22FFFFFFFFD	2020-12-23	2020C23FFFFFFFFD	P1D	FFFFF01FFFFFFFFC
C	2019-01-12	2019112FFFFFFFFD	2019-06	20196FFFFFFFFFFD	P4M19D	FFFF419FFFFFFFFC
D	2018	2018FFFFFFFFFFD	2021-01	20211FFFFFFFFFFD	P3Y1M	00031FFFFFFFFFC
E	2015-05	20155FFFFFFFFFD	2019	2019FFFFFFFFFFD	P3Y7M	00037FFFFFFFFFC

Example 10: Taking Advantage of the Default Substitution to Calculate ISO Duration Values

The columns are shown with and without formatting. As you can see, the values in AESTDT and AEENDT are stored as SAS ISO 8601 values, which are stored in character variables. I cannot stress this enough, because we're used to dealing with dates and times as numbers in SAS. Another difference from regular SAS dates and times is that all three variables use the $N8601E. format, but they display differently. When you use this format, SAS figures out the context from the values and chooses the correct display.

Let's look at the rows where SAS has used its default substitutions. Row A had a missing end date. Without a year in the data, or specified in the substitution parameter, SAS does not do any substitution. Note the missing end date in this row displays as asterisks, not blanks, but the duration is blank. Row C substitutes the value 1 for the missing day in the end date. Row D substitutes 1 for the start month and day, and end day, while row E does the reverse, substituting 1 for start day, and 1 for both end month and day. Rows B through E substitute 0 for hours, minutes, and seconds. You could redefine the default substitutions if desired, but remember, it is always a simple component substitution, not an imputation.

Another use of the IS8601_CONVERT routine is to easily convert ISO durations into their SAS time/datetime values. The variable HOURS is numeric, but the ISO duration is character. The first two parameters in the call (line 5) are 'du' because you are taking a duration and turning it into a duration. Example 11 illustrates:

```
1.   DATA duration_convert;
2.   INFILE datalines PAD MISSOVER;
3.   LENGTH aedur $ 32 hours 8;
4.   INPUT @1 aedur :$N8601E32.;
5.   CALL IS8601_CONVERT('du','du',aedur,hours);
6.   aedur_unf = aedur;
7.   hours_unf = hours;
8.   FORMAT aedur $N8601E16. hours time8.;
9.   DATALINES;
10.  P1D
11.  P1M2D
12.  PT14H23M
13.  P3M12DT6H30M
14.  ;
15.  RUN;
```

	Duration of Adverse Event	Unformatted Duration of Adverse Event in ISO Data Format	Duration of Adverse Event (h)	Duration of Adverse Event (sec)
A	P1D	FFFFF01FFFFFFFFC	24:00:00	86400
B	P1M2D	FFFF102FFFFFFFFC	768:00	2764800
C	PT14H23M	FFFFFFF1423FFFFC	14:23:00	51780
D	P3M12DT6H30M	FFFF3120630FFFFC	2454:30	8836200

Example 11: Getting SAS Time Values from ISO Durations

As you can see, the ISO duration has been converted to a SAS time value without having to derive a start or end date. Example 12 shows how to calculate the end SAS datetime using this routine if you have an ISO start date and an ISO duration. The only difference from Example 10 is in the first parameter ('dt/du') in the CALL IS8601_CONVERT, which says we're working with a date-time (or datetime) and an ISO duration:

```
DATA calc_enddate;
INFILE datalines PAD MISSOVER;
LENGTH aestdt $ 32 aedur $ 32 enddate 8;
INPUT @1 aestdt :$N8601E. @12 aedur :$N8601E.;
CALL IS8601_CONVERT('dt/du','end',aestdt,aedur,enddate);
aestdt_unf = aestdt;
aedur_unf = aedur;
enddate_unf = enddate;
FORMAT aestdt aedur $N8601E16. enddate dtdate9.;
DATALINES;
2016-04-05 P6D
2017-02-15 P1M11D
2016-11-17 P3W
2014-08-22 P3M12D
;
RUN;
```

Start Date of Adverse Event	Unformatted Start Date of Adverse Event	Duration of Adverse Event	Unformatted Duration of Adverse Event	Calculated End Date of Adverse Event	Unformatted Calculated End Date of Adverse Event
2021-04-05	2021405FFFFFFFFD	P6D	FFFFF06FFFFFFFFC	11APR2021	1933718400
2021-02-15	2021215FFFFFFFFD	P1M11D	FFFF111FFFFFFFFC	26MAR2021	1932336000
2019-11-17	2019B17FFFFFFFFD	P3W	000000000000030E	08DEC2019	1891382400
2020-08-22	2020822FFFFFFFFD	P3M12D	FFFF312FFFFFFFFC	04DEC2020	1922659200

Example 12: Calculating Event End Dates from Start Date and ISO Duration

The large numbers in the last column are SAS datetime values representing seconds since midnight, January 1, 1960. The DTDATE. format has been used to display them as a date in the table above, but it is a SAS datetime value. The time components have been set to zero through the default substitution in the IS8601_CONVERT routine.

This is just an overview of some of the tasks that can be automated easily with the IS8601_CONVERT routine. For more details and possibilities, consult the SAS documentation.

TIME ZONE AND UTC

Are you using SAS 9.4? If so, then you can specify the time zone in which you are located. By default, this is the value set in the TIMEZONE system option (which may have been set by your SAS administrator.) Unlike other system settings, the LOCALE= system option does not automatically define a time zone, as there may be several time zones within a given locale. (Trivia question: how many *time-zone-id*s are in the US? Sixty-seven, and Indiana is responsible for twenty-two of them! The US spans 7 hours across those 67.) You can find out what time zone has been set for your SAS session by using the TZONEID() function:

```
serverTimeZone = TZONEID();
```

If the TZONEID () function returns a blank, then it has not been set, and you will have to do it using the TIMEZONE system option as shown below. You cannot use the three- or four-letter abbreviations we tend to think of as time zone. For example, CST can stand for Central, China, or Cuba Standard Time. You will have to consult the SAS documentation to find the valid *time-zone-ID* value you want. The values are java-compatible, but there are over 500 of them worldwide.

```
OPTIONS TIMEZONE = time-zone-ID;
```

Once the TIMEZONE option has been set, you can use the TZONENAME() function to get the time zone acronym (CST, EDT, etc.), including any daylight savings time adjustment, if applicable:

```
myTimeZone = TZONENAME();
```

Of course, there are formats and informats that interact with the TIMEZONE option, as well as functions that calculate the offset from UTC. You should be aware that the offset is not a whole hour in some locations, so always make sure you include minutes when you display the offset value. If you need to work with time zones and offsets from UTC, consult the SAS documentation for National Language Support (“NLS.”)

IS0 8601 IN R

R provides native support for working with dates and times that align closely with the ISO 8601 standard. In R, date and time values are represented using specialized object classes that store temporal information as numeric values with associated attributes. These classes allow for consistent representation, arithmetic operations, and formatting of date and time data. The most used object classes in R are Date and POSIXct. Date and datetime values are stored as numeric values with associated class attributes. While SAS stores dates and times as numeric values relative to 1960-01-01, R uses a similar numeric representation based on the 1970-01-01 epoch.

The Date class is used to represent calendar dates without a time component, while POSIXct is used to represent date-time values that include both date and time. Because ISO 8601 formats follow structured representations such as YYYY-MM-DD and YYYY-MM-DDThh:mm:ss, they align naturally with these classes and can be interpreted by R with minimal specification.

BASE R DATE-TIME PARSING AND FORMATTING

In base R, character values are converted to date or datetime objects using functions such as as.Date() and as.POSIXct(). As shown in Example 13, when checking the data type after conversion using as.Date(), R stores these values as numeric (double) representations corresponding to time elapsed since a specified epoch.

```
> as.Date("2026-03-06")
[1] "2026-03-06"
> class(as.Date("2026-03-06"))
[1] "Date"
> typeof(as.Date("2026-03-06"))
[1] "double”
```

Example 13: Date Representations in R

Parsing datetime values can introduce subtle issues. As shown in Example 14, when attempting to parse an ISO 8601 datetime containing the "T" separator, the resulting output may not retain the expected time component.

```
> as.POSIXct("2026-03-06T14:35:00")
[1] "2026-03-06 UTC"
```

Example 14: POSIXct “T” Separator Behavior

Although the input clearly includes a time value, the output only displays the date. This behavior is related to the ISO 8601 "T" separator, which distinguishes the date and time components (e.g., YYYY-MM-DDThh:mm:ss). While this format is standard in ISO 8601 and commonly used in clinical datasets, base R does not always interpret the full datetime string as intended unless the structure is explicitly defined.

```
> as.POSIXct("2026-03-06T14:35:00",format = "%Y-%m-%dT%H:%M:%S")
[1] "2026-03-06 14:35:00 UTC"
```

Example 15: Datetime Representations in R

To address this, the format argument can be used to specify the expected structure of the input. In Example 15, providing a corresponding strftime format ensures that both the date and time components are correctly parsed. After proper parsing, R displays datetime values using a space rather than the "T" separator. This is a display convention and does not affect the underlying value.

As shown in Example 15, parsing and formatting in base R relies on strftime format. Table 2 summarizes commonly used specifiers.

Specifier	Description	Example
%Y	Four-digit year	2026
%m	Month	03
%d	Day of month	06
%H	Hour	14
%M	Minute	35
%S	Seconds	01

Table 2: Strftime Formats

The format() function applies strftime format specifications to date and datetime objects to produce character representations of those values. Unlike SAS formats, which alter display without changing the underlying type, the format() function in R returns a character vector. As shown in Example 16, applying format() converts a POSIXct object into a character representation.

```
> dt <- as.POSIXct("2026-03-06 14:35:00")
> formatted_dt <- format(dt, "%Y-%m-%dT%H:%M:%S")
> formatted_dt
[1] "2026-03-06T14:35:00"
> class(formatted_dt)
[1] "character"
> class(dt)
[1] "POSIXct" "POSIXt"
```

Example 16: R Datetime Formatting Using format()

Recall that SAS uses 1960-01-01 as its epoch, while R uses 1970-01-01. As shown in Example 17, this difference can lead to discrepancies when converting numeric values to dates.

```
> x <- 18262
> as.Date(x, origin = "1970-01-01")
[1] "2020-01-01"
> as.Date(x, origin = "1960-01-01")
[1] "2009-12-31"
```

Example 17: SAS vs R Epoch Difference

While Base R provides flexible tools for parsing and formatting, the need to explicitly define formats and manage type conversions can introduce additional complexity. The lubridate package simplifies these operations by providing a more intuitive grammar for working with dates and datetimes.

PARSING DATES AND TIMES USING LUBRIDATE

The lubridate package provides a streamlined approach to parsing date and datetime values in R, particularly for data that follow or resemble the ISO 8601 standard. While base R often requires explicit format specifications for parsing operations, lubridate introduces a grammar-based approach that allows date and datetime values to be interpreted directly from their representation.

A key advantage of lubridate is its flexibility. Parsing functions can accommodate variation in punctuation and separators while returning a standardized ISO 8601 representation. For example, the R function mdy() can interpret multiple month-day-year representations and return a Date object:

```
library(lubridate)

x <- c(
       "MAR062026",
       "03.06.2026",
       "03-06-2026",
        "03062026"
       )

mdy(x)
```

```
[1] "2026-03-06" "2026-03-06" "2026-03-06" "2026-03-06"
```

Example 18: Flexible Evaluation of mdy( )

As shown in Example 18, differences in separators and formatting do not affect the resulting value. Lubridate normalizes these inputs into a consistent ISO 8601 calendar date representation. This improves readability and reduces the need for explicit format handling compared to base R.

Lubridate parsing functions follow a consistent naming convention based on the order and inclusion of date-time components. Rather than enumerating all available parsing functions, the naming structure allows the user to infer the appropriate parser for a given input. Table 3 illustrates this concept.

Component Pattern	Meaning	Example Function	Example Input
y,m,d (order defines structure)	Year, Month, Day in specified order	ymd(), dmy()	“2026-03-06”, “06.03.2026”
_h, _hm, _hms	Adds time components to date	ymd_hms()	“2026-03-06T14:35:00”
Combined patterns	Date + time parsing with flexible separators	dmy_hm()	“06-03-2026 14:35”

Table 3: Lubridate Parsing Function Structure

The function name encodes both the order of date components and, when present, the time components through suffixes such as _h, _hm, and _hms. This eliminates the need to explicitly define format strings as required in base R.

Datetime parsing follows the same pattern. For partial datetime values, the parser should match the level of precision present in the input:

```
> ymd_hms("2026-03-06T14:35:20")
[1] "2026-03-06 14:35:20 UTC"
> ymd_hm("2026-03-06T14:35")
[1] "2026-03-06 14:35:00 UTC"
> ymd_h("2026-03-06T14")
[1] "2026-03-06 14:00:00 UTC"
```

Example 19:Parsing Datetime Partials

In Example 19, the "T" separator does not affect parsing. Lubridate correctly interprets ISO 8601 datetime structures without requiring explicit format definitions.

In practice, datasets often contain heterogeneous date representations. While it is possible to conditionally apply different parsing functions, lubridate provides a more scalable solution through the parse_date_time() function, which evaluates multiple candidate formats within a single call. Consider Table 4, where both AE start and end dates contain mixed formats.

AESTDTC	AEENDTC
2015-08-12	2015-10-10T05:33:51
2013-05-03T14	2013-05-04T16
03/09/2020T13:01	2020-04
22-01-2023T12:13:40	23JAN2023
2014	12-01-2014T20:22:10
March 05, 2008 19:20:11	2008-03-06

Table 4: Heterogeneous AE Start and End Dates

By passing a vector of different date formats like lubridate parsing functions, we can use the parse_date_time() function. Below is a tidyverse example of calling parse_date_time().

```
library(dplyr)
library(lubridate)

orders_vec <- c(
  "ymd",
  "ymd H",
  "ymd HMS",
  "mdy HM",
  "dmy HMS",
  "dby",
  "ym",
  "y",
  "B d, y HMS"
)

ae_dates_parsed <- ae %>%
  mutate(
    AESTDTM  = parse_date_time(AESTDTC, orders = orders_vec),
    AEENDTM  = parse_date_time(AEENDTC, orders = orders_vec)
  )
```

AESTDTC	AEENDTC	AESTDTM	AEENDTM
2015-08-12	2015-10-10T05:33:51	2015-08-12 00:00:00	2015-10-10 05:33:51
2013-05-03T14	2013-05-04T16	2013-05-03 14:00:00	2013-05-04 16:00:00
03/09/2020T13:01	2020-04	2003-09-20 20:13:01	2020-04-01 00:00:00
22-01-2023T12:13:40	23JAN2023	2023-01-22 12:13:40	2023-01-23 00:00:00
2014	12-01-2014T20:22:10	2014-01-01 00:00:00	2014-01-12 20:22:10
March 05, 2008 19:20:11	2008-03-06	2008-03-05 19:20:11	2008-03-06 00:00:00

Example 20: Homogeneous AE Start and End Date/Times

Following parsing, all values are standardized into ISO 8601-compliant datetime representations. Partial dates are imputed to the earliest valid time point (e.g., "2014" - "2014-01-01 00:00:00"), and the resulting variables are stored as POSIXct objects. While this behavior ensures that a complete datetime value is

returned, it may not align with application-specific imputation rules or analytical requirements. As such, users should avoid relying on default imputation when deriving variables and instead apply controlled, context-appropriate imputation logic.

Despite this flexibility of parse_date_time(), care must be taken when working with ambiguous date formats. For example, in Table 4, the value "03/09/2020T13:01" is inherently ambiguous. In Example 20, parse_date_time() incorrectly parsed this value as "2003-09-20 20:13:01". This occurs because the function evaluates candidate formats and selects a structurally valid match, even if the interpretation is incorrect. When multiple candidate formats are supplied, the order and selection of those formats can influence the result. As such, incorrect parses may occur silently without generating errors. This highlights the importance of specifying appropriate and unambiguous parsing orders when working with heterogeneous datetime data.

SUMMARY

The ISO 8601 standard has been adopted by CDISC as the standard for dates and times in clinical studies, which makes an understanding vital for clinical SAS programmers. However, this standard is global, and it has applications far beyond the realm of CDISC standards. The standard accommodates missing components in its date, time, and date-time values. Unfortunately, this is incompatible with the way we think about SAS dates, times and datetimes, which need to point to an exact moment so an exact value can be assigned. Without this exact value, SAS will leave it missing, possibly leading to a loss of information.

The E8601DT. and E8601DA. informats reliably convert extended notation ISO 8601 date-time and date values to SAS without needing to parse the ISO string. Similarly, to display a SAS date or datetime value in its ISO string representation, you can use the E8601DT., or the E8601DA., or the E8601DN formats. If you need to read/write time on its own, the TIME5. format and informat are ISO 8601-compliant. The ISO 8601 standard for times includes time zone information, which is often unnecessary or unavailable in our clinical programming environment.

Even though normal date, time and datetime handling in SAS requires an exact value, there is a fully compliant ISO 8601 facility in SAS, and it revolves around the IS8601_CONVERT routine. This is a call with no returned value. Values read are stored in character variables with an internal data format unique to ISO 8601 data, much like SAS does with standard dates, times, and datetimes. As with standard SAS dates, times and datetime, these ISO data need formats to understand them, and informats to convert ISO strings to SAS values. The IS8601_CONVERT routine is capable of much more than creating ISO data in SAS. Calculating durations and displaying them in ISO 8601-compliant form is a prime example, replacing the need to break down a SAS datetime value in seconds into years, months, days, hours, minutes and seconds, and then assemble the ISO duration string with concatenation.

In contrast, R provides native support for ISO 8601-compliant date and datetime representations through its Date and POSIXct classes, which align naturally with the structure of ISO 8601 strings. Packages such as lubridate further simplify the parsing and manipulation of ISO 8601 data by allowing date-time values to be interpreted directly from their representation without requiring explicit format specifications. This enables efficient handling of both complete and partial ISO 8601 values and supports flexible workflows when working with heterogeneous date formats commonly encountered in clinical data.

The examples in this paper should help improve your understanding of the ISO 8601 standard and improve your efficiency in your day-to-day clinical programming involving the standard.

REFERENCES

Morgan, Derek P. 2014. *The Essential Guide to SAS® Dates and Times, Second Edition*. Cary, NC: SAS Institute Inc.

SAS Institute Inc. 2016. SAS® 9.4 National Language Support (NLS): Reference Guide, Fifth Edition. Cary, NC: SAS Institute Inc.

Richann Watson, Worried about that Second Date with ISO®? Using PROC FCMP to Convert and Impute ISO 8601 Dates to Numeric Dates, PharmaSUG 2025 Proceedings, https://www.lexjansen.com/pharmasug/2025/AP/PharmaSUG-2025-AP-002.pdf, 2026-03-27

Grolemund, Garrett, and Hadley Wickham. 2011. Dates and Times Made Easy with lubridate. Journal of Statistical Software 40(3): 1–25. https://www.jstatsoft.org/v40/i03/, 2026-03-27

ACKNOWLEGEMENTS

As always, Derek would like to thank SAS Technical Support for all the help they've given him over the course of his career.

Marckenley would like to thank the BMS Hematology, Oncology, and Cell Therapy Statistical Programming organization for their support and for fostering an environment that encourages innovation and continuous learning.

CONTACT INFORMATION:

Further inquiries are welcome to:

Derek Morgan
E-mail: derek.morgan@bms.com OR mrdatesandtimes@gmail.com

Marckenley Mercie
E-mail: marckenley.mercie@bms.com OR marckenley.mercie789@gmail.com