Paper 4934-2020

Getting the time right across time zones

Frank Poppe, PW Consulting

ABSTRACT

Suppose you enter in your digital agenda the date and time for a presentation in, say, Washington, while living in another time zone. When in Washington you look at your agenda and wonder: in what time zone is the time displayed? The local time at home, or the current time?

Those kind of problems occur also when computers in different time zones work together, or when daylight savings time (DST) is involved.

A common practice to solve this is to store all time information in UTC-time: Universal Coordinated Time, commonly known as Greenwich Mean Time (GMT).

SAS® has several functions and formats that translate UTC time to local time and vice versa, but there is still room for misunderstandings.

SAS has to know the local time zone, and there are two ways of defining them. One of those take DST into consideration, the other does not.

The formats produce long strings, often too long to use as tick marks in graphs. Using the 'directives' of the FORMAT procedure is not an option: they do not translate from UTC, and they cannot be used in ODS Graphics.

Another issue is that countries can change the rules for DST. Maroc suddenly made DST permanent in 2018, and the EU might abolish it in 2021. The SAS functions have the rules hardcoded.

To overcome this we would need some logic inside a format, but that is not something PROC FORMAT provides. But: the FMCP procedure allows us to define functions, and PROC FORMAT allows to use functions. This paper will describe this method.

Code for the functions is included.

INTRODUCTION

SAS has many functions that enable users to manipulate datetime values, converting from UTC time to local time, and to display them in the proper locale.

Suppose you enter in your digital agenda the date and time for a presentation in, say, Washington, while living in another time zone. When you are in Washington you look at your agenda, and wonder in what time zone that time is being displayed. The local time at your home, or your current local time?

Those kind of problems occur also when computers in different time zones work together, or only when summer and winter time is involved.

A common and sound practice is to store all time information in UTC-time: Universal Coordinated Time. This is, for all practical purposes, Greenwich Mean Time. GTM, the standard time in Greenwich near London, on the zero meridian, without anything like summer time.

SAS has several functions and formats that translate UTC time to local time and vice versa, but there is still a lot of room for misunderstandings.

When applying those formats and functions SAS has to know the local time zone, and there are two ways of defining them. One of those take summer and winter time into consideration, the other does not.

The formats produce in general long datetime strings with full date, time and time zone indications, like 17Mar2013:19:30:22 +0500. These are often too long to use in graphs as indicators on axes. Proc Format has the feature of 'directives', enabling users to specify exactly which parts to use. But the translations between UTC and local time is lost, and - more important- formats using the directives cannot be used in ODS Graphics.

To overcome this we would need some logic inside a format, but that is not something Proc Format provides. But: Proc FMCP allows to define functions, and Proc Format allows to use functions. This paper will describe this method.

TIME, DATE AND DATETIME IN SAS: THE BASICS

In SAS you can specify a date, a time and a datetime. But essentially they are all numbers - as any element in a computer language or a database is in the end just one or more bytes with a numerical value. The definition of these three concepts is as follows.

DATE

A date is defined as the number of days since January 1, 1960, which is day 'zero'. Day 1 is January 2, 1960 and day -1 is December 31, 1959.

The documentation states that SAS can handle dates up to the year 19,900. Some experiments show that the date functions seem to work correctly up until December 31, 20,000. For larger values they return an error.

Note that the standard SAS formats only show the last four digits of the year value.

TIME

Time is defined as the number of seconds since midnight. Thus a time value of +1 is 00:00:01, and +60 is 00:01:00. Negative values are interpreted as if one is counting back from midnight, thus -1 is 23:59:59.

When ordering observations with negative and positive values for time they are ordered just as any number, but in displaying there is no difference between a value of -1 and a value of 84239. Both are displayed as 1 second before midnight: 23:23:59.

DATETIME

Datetime values count the number of seconds since January 1, 1960, 00:00:00, which is represented by the value 0 (zero). So January 1, 1961, at 00:00 is the value 31,622,400 which is 365*24*60*60. And January 1, 1963 is 94694400, which is
(3*365 + 1) * 24*60*60.
Note the extra day, because a leap year is involved.

The documentation does not specify what the largest positive value is that SAS can handle for datetime values, but, just like the date functions, they seem to work correctly up until the end of the year 20,000, i.e. the number 569,318,630,399 (the last second of the year 20,000).

This value, in scientific notation 5.7e11 ($5.7 \cdot 10^{11}$), is still much less than the largest integer number SAS can represent exactly. That is (on Windows and Unix) 9,007,199,254,740,992 or 9.0e15.

FRACTIONS

SAS does not attribute a 'meaning' to fractions in date values. Fractions are ignored when a day is displayed using a SAS format. That is, arithmetically the FLOOR function is applied, meaning that a value of +10.6 is processed like +10, and -10.6 as -11.

An application of course can attach a meaning to a fraction, implying that something that happens on day 10.6 happened somewhere past noon, i.e. at January 11, 1960, 14:24:00, but for SAS to interpret it in that way one has to convert it to a datetime value by multiplying 10.6 with 24 * 60 * 60.

In time and datetime values the fraction just is what it is: a fraction of a second. That is why the standard SAS formats for time and datetime have a .d specifier in the syntax, while the date formats have not.

TIME ZONES

The world is divided in different time zones. Many countries only have a single time zone, but if the country stretches out wide enough in the east-west direction, there often are different time zones. To have a common understanding of 'time' across the world the UTC standard time is being used. 'UTC' is short for Universally Coordinated Time or (in French) Temps Universel Coordoné; the abbreviation UTC is a kind of negative compromise as the abbreviation does not fit for either language. And it does fit in with earlier variants of a universal time under the names of UT0, UT1, etc.

The UTC time is equivalent to the time at the Greenwich zero meridian, GMT (accept for a sporadic temporary second per year to account for deviations in the rotation of the earth).

Figure 1. Greenwich Observatory, at 0º

Theoretically the world could do with 24 time zones: UTC, UTC+1 to UTC+11 and UTC-1 to UTC-11, and an arbitrary choice for the 24th: UTC-12 or UTC+12. But there are more. One reason being that some areas opt for a time difference in not whole hours.

And both time zones UTC-12 and UTC+12 are in use, and some areas even have UTC+13 and UTC+14 because they want their clocks synchronized with the area they are politically or otherwise attached to. This means there are islands (in the Pacific Ocean), lying on more or less the same meridian, with on one island the clock a full day plus 2 hours ahead of the other island.

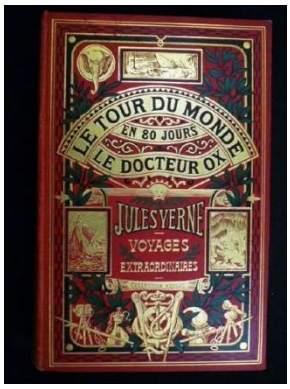


Figure 2. Jules Verne

More room for misinterpretation has been added by the concept of Daylight Saving Time (DST), also known as summer time. Not only does that mean that the difference between local time and UTC changes two times per year, but some areas close to each other do observe DST while others do not, and areas can decide to change that, and the starting and ending moments of DST vary between areas.

Nowadays many people and organizations interact across the world, and consequently their applications and data interact as well. So it is important to know with respect to which time zone a date and a time have been recorded in order to interpret them correctly again. (For how things can go wrong, or miraculously go right, read Jules Verne's "Around the World in Eighty Days" (published in 1872 as "Le tour du monde en quatre-vingts jours").

ISO 8601 STANDARD

The international UTC standard deals only with the time itself, the value, not with the way that value is being represented in characters and language. Does 7/8 mean July 8 or August 7? Is 6 o'clock in the morning or in the afternoon? Etcetera.

To remove also these kind of ambiguities in the representation of date, time and datetime values the ISO organization developed the ISO 8601 standard. It also deals with durations, for which SAS also has appropriate functions, informats and formats, but those are not discussed in this paper.

The basis of the standard is that values are notated with all elements numeric, in order from large to small. So a datetime value starts with the year (4 digits), then the month, etcetera, ending with seconds and fractions of seconds. Dates have only year, month and day, time obviously only hour, minutes and seconds.

Year should always have four digits, the other elements always 2 digits (so values smaller than 10 are preceded by a 0 (zero).

In a datetime value the date and the time are separated by a T.

In the extended notation all date elements are separated by a – (dash) and the time elements by : (colon). In the basic notation the separators are omitted. In the ISO standard then also the T-separator may be omitted, but in SAS it always must be present.

In time and datetime values a time zone indicator can be added, with a plus or minus and the difference with respect to UTC in hours and minutes (the minutes usually being 00). The zero time zone can be indicated in three ways: +00:00, or –00:00, or simply with Z.

There are also rules for dates before 1583 (when the Gregorian calendar starts) and beyond 9999, but they are not relevant here. The same is true for the standards for the notation of week numbers (0 through 53) and day numbers (1 through 366).

As an example, the datetime value for May 11, 1954 at 9 hour, 30 minutes and 40.03 seconds, which in SAS is the numeric value of -178036159.97, will show up in extended ISO 8601 notation as 1954-05-11T09:30:40.03.

If this is a UTC time, and one would want to see it as the local time in a time zone one hour ahead of UTC, this would show as 1954-05-11T10:30:40.03+01:00.

One way to remove any uncertainty would be to store all values as character values using the ISO notation including the time zone indication, like "1954-05-11T10:30:40.03+01:00" or "19540511T10304003+0100" in the basic notation.

But in SAS then we would lose all the possibilities of manipulating the values as a number, and comparing different values would become cumbersome and tricky.

A good alternative would be to store all values as UTC values. This then requires that on input adjustments might be needed to change a local time or datetime value into UTC. And on output, when external reports are produced, adjustments might be needed as well, depending on requirements.

For those adjustments on input and output SAS has a number of ISO-formats and informats, which shall be discussed in a later paragraph.

NATIONAL LANGUAGE SUPPORT (NLS)

Under the heading of National Language Support (NLS) SAS brings together two related but also different concepts:

- Internationalization.
- Localization.

This paper focuses primarily on internationalization, and within that on the time aspect.

Internationalization is commonly referred to as I18N (the word starts with an I and ends with an N, with 18 characters in between). Its purpose is to make code independent of local settings. One of those settings is the **TIMEZONE** system option. Another important aspect of I18N is the encoding and character set used.

Localization mainly involves the way data is read on input, and how it formatted and presented on output. This is defined by the setting of the **LOCALE** system option. Localization only will be touched upon in this paper.

NL AND ISO FORMATS AND INFORMATS

The traditional date, time and datetime formats like DATE., time and datetime have since long been supplemented with some formats for specific languages and locales, like Japanese (e.g. JNENGO.) and Hebrew (e.g. HEBDATE.).

This now has been generalized in the NL formats and informats (all names start with NL) and the ISO formats (following the ISO 8601 convention), the names all starting with B8601 or E8601 (Basic or Extended).

The ISO informats and formats read and write date, time and datetime values that are formatted according to ISO 8601. If the proper variant of the formats is used, they will also read or write the timezone, and some variants also provide automatic adjustment of the value, between the timezone specified in the formatted string, the TIMEZONE setting or the UTC time. Exactly which variant does which adjustment will be described in a later paragraph.

The NL formats and informats have only to do with localization; they do no interact in any way with a TIMEZONE setting, only with the LOCALE setting. They produce, resp. consume, local text strings that represent a date, time or datetime value in the current locale. This involves things like the name and the order of date elements.

These NL formats will not be described in this paper, except for the following general remarks on the documentation.

The general description of all NL-informats is "Reads the value in the specified locale and then converts the date value to the local value". This is rather misleading, as no 'conversion' takes place, it just reads the value according to the locale, producing the SAS date, time or datetime value.

Also confusing is that the documentation on the informats contain many phrases that are only relevant to the corresponding format, like "specifies the width of the output field. If necessary, SAS abbreviates the date to fit the format width". What should be stated there is that it specifies the width of the input field, and that an abbreviation of the month is allowed.

For the NL-formats the general description is "Converts a SAS date value to the date value of the specified locale, and then writes the date value as a date". Again there is no conversion, the format just formats the values in the current locale. And the current locale is not 'specified' in the format, it is the current value of the system option LOCALE.

TIME ZONE ID'S AND NAMES IN SAS

The TIMEZONE option can take two types of values an Id or a Name. The definitions are (taken literally from the documentation):

- "Time-zone-name: specifies a three- or four-character time zone name."
- "Time-zone-id: specifies a region/area value that is defined by SAS. When you specify a time zone ID, the time zone that SAS uses is determined by time zone name and daylight-saving time rules."

Thus a time-zone-name is a string like CET, CEST, EST, EDT, etc. - and thus looks more like an id-field…

The time-zone-id's are more meaningful strings: Europe/Amsterdam, or America/New_York – and thus look more as a name…

All possible values can be found in the appendix to the documentation of the SAS® 9.4 National Language Support (NLS): Reference Guide, Fifth Edition. The table lists the time-zone-id's under the heading 'Time Zone Information'.

DAYLIGHT SUMMER TIME (DST) OR SUMMERTIME

The definition above of the two terms has an important difference that can easily be overlooked: using the Id mean that DST is automatically taken into account.

This means that the difference between UTC and local time will be one hour larger if, according to the area indicated by the Id and the current date and time, DST is in effect. If a time-zone-name is used, like EST (Eastern Standard Time) the user implicitly specifies that never MET (Eastern Daylight Time) should be used. Specifying the Id America/New_York on the other hand will take that into account.

DATE AND TIME FUNCTIONS

The DATA Step Date() and Today() functions give the current date as a SAS numeric value, and Time() gives the current time. Datetime() gives the datetime value.

Note that if the TIMEZONE option has been set, a difference between the timezone set for the system and the timezone option for SAS will influence the result of these functions. SAS will adjust the date and time of the system clock to the timezone that is currently in force in the SAS session.

THE ISO INFORMATS AND FORMATS

There are about 20 formats in this category, but the description of how they work can be summarized as follows.

For each function there is a basic and an extended version, the name starting with B8601 or E8601. They operate in the same way, except that the extended versions accept or write delimiters between the elements. The basic versions only expect or write a 'T' between the date and time parts.

There are two informats and formats that read or write date values. The B6601DA and E8601DA variants read and write date values, while the B8601DN and E8601DN variants select the date from a datetime value.

The other informats and formats fall into 4 groups, each group having a B8601 and an E8601 version, and each having a format for time and for datetime values. The groups differ in the way they read or write a time zone indicator, and whether they adjust value for a difference between a timezone in the value, the timezone in effect in the SAS session, or the UTC.

The following table show how the different informats process the values that are read.

Informats B8601*xx* and E8601*xx*	**Two character suffix *xx***	
	Time	**Datetime**
Store value as is	TM	DT
Adjust value read to UTC, according to time zone indicator (must be present)	TZ	DZ
Adjust value from indicated time zone to local (indicator Z not allowed, if time zone indicator is not present, no adjustment)	LZ	LX
Adjust value to UTC (time zone must be present).	TX	DX

Table 1. Overview of the ISO informats

For formats the behavior is more or less similar.

Formats B8601*xx* and E8601*xx*	**Two character suffix *xx***	
	Time	**Datetime**
Display value as is, no time zone indicator	TM	DT
Display value as is, add zero time zone indicator (Z or +00:00)	TZ	DZ
Display value as is, add time zone indicator according to TIMEZONE setting	LZ	LX
Adjust value from UTC to local time according to TIMEZONE setting, and add time zone indicator	TX	DX

Table 2. Overview of the ISO formats

For both tables this is the result from experiments in SAS 9.4, maintenance releases M3 and M6. These releases produce the same results, with one small exception. The M3 release does not quite correctly determine the hour at which DST comes into effect and ends for the European Union. It puts both at 1 o'clock a.m. local time, but the correct way is at 1 a.m. **UTC***, for all time zones* in the EU (unlike the USA, where the change takes place at 2 a.m. local time at each time zone). Somewhere between M3 and M6 this has been corrected.

Note also that the descriptions above are not always what I expected while reading the documentation, but this is what I observed running the code.

PRACTICAL PROBLEMS

Whether or not an ISO format is being used that adjusts the datetime to local time, the string produced by the format is always fairly long, as it always includes the full date and time information. The other SAS formats, both the original ones and the NL-versions, have variants that display only a single or a few aspects of the complete datetime value (e.g. only the date, or only the time, or only the name of the month or the day of the week, etc.).

This paper originated when the author was working on a project that involved long series of measurements, for which the time was stored as UTC datetime values. From these series separate time slices had to be presented in tables and graphs. These time slices covered ranges of a single day to one week.

Within those ranges the ISO format produces a lot of redundant information, repeating year and month, plus the 6 characters extra for the time zone indicator ("-01:00") which led to

awkward and visually not attractive tables and graph. The following examples illustrates that.

USING EXISTING FORMATS

For this, and in later examples, the following data series is created.

```
data test (keep = UTCdt y) ;
pi = constant ('pi') ;
step = 60 ;
do UTCdt = '1jan2020:00:00'dt to '1jan2021:00:00'dt by step ;
    t = timepart ( UTCdt ) ;
    d = datepart ( UTCdt ) ;
    rad = 2 * pi * t / 3600 ;
    fuzz = ranuni ( 123454321 ) ;
    y = d + fuzz + t/86400 + sin ( rad )  ;
    output ;
end ;
run ;
```

When the datetime values have to be displayed as local values, adjusted for the local time zone, the ISO formats can be used.

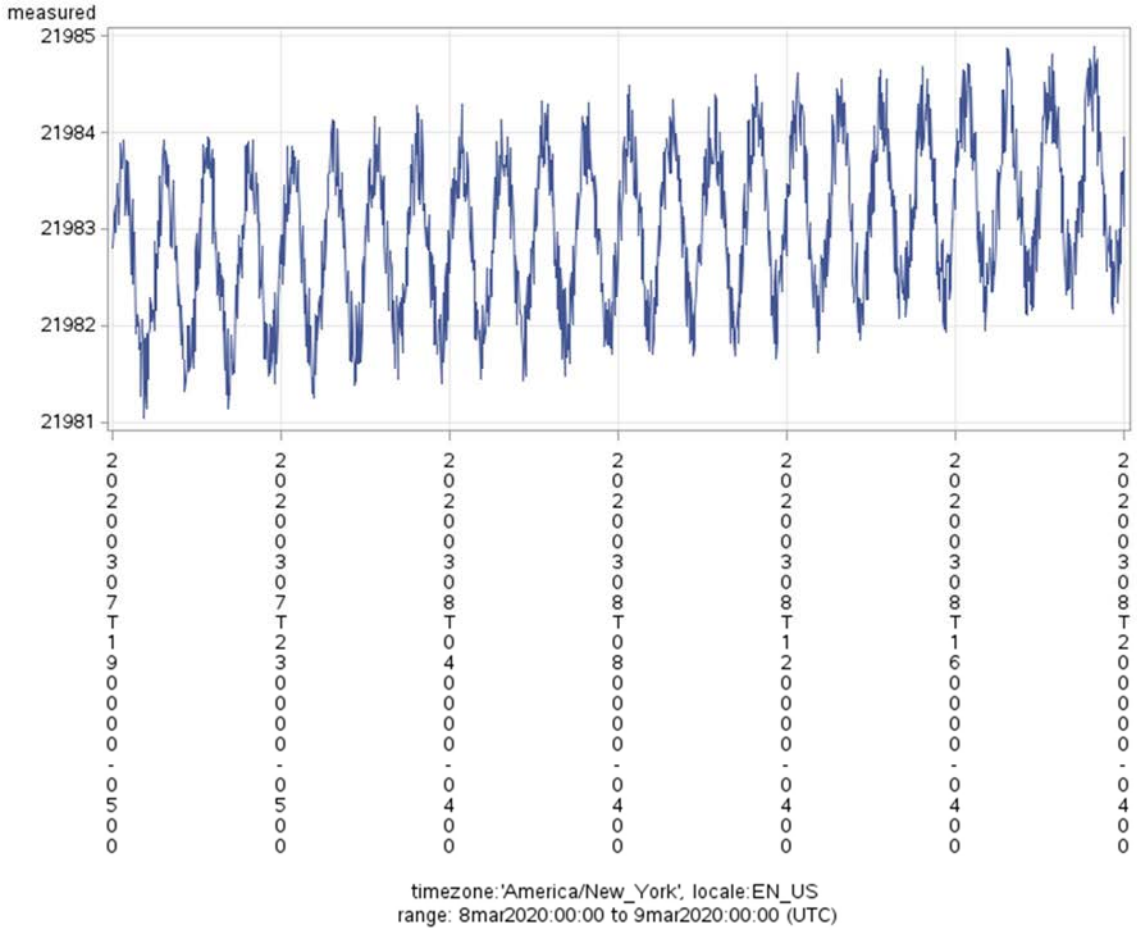
Using the REPORT Procedure, a few copies of the UTCdt column are displayed, with different formats. The first column show the 'raw' UTC value, formatted for the current locale.
The two next column show the basic and extended version of the DX-variant of the ISO family of formats. Note that date and time have been adjusted for the difference between the local time zone and UTC: 2 hours ahead, during DST. But they need a lot of room for that. The last two columns show the result when the formats are being used that extract the time part. (The PROC REPORT code is in the appendix.)

timezone:'Europe/Amsterdam', locale:NL_NL

	ISO8601 DX		timepart	
UTC, NL format	Basic	Extended	standard	NL format
01 april 2020 00:00:00 uur	20200401T020000+0200	2020-04-01T02:00:00+02:00	00:00:00	00:00:00 uur
01 april 2020 00:01:00 uur	20200401T020100+0200	2020-04-01T02:01:00+02:00	00:01:00	00:01:00 uur
01 april 2020 00:02:00 uur	20200401T020200+0200	2020-04-01T02:02:00+02:00	00:02:00	00:02:00 uur
01 april 2020 00:03:00 uur	20200401T020300+0200	2020-04-01T02:03:00+02:00	00:03:00	00:03:00 uur
01 april 2020 00:04:00 uur	20200401T020400+0200	2020-04-01T02:04:00+02:00	00:04:00	00:04:00 uur

Display 1. Table showing result ISO formats

The GPLOT Procedure has been used to extract 24 hours of the test data, and graph the y variable against the datetime, which is formatted using the B8601DX. Format. Because of the length of the string, the values have to be rotated by PROC GPLOT.



Display 2. Graph using the B8601DX format

The time slice has been chosen deliberately so that the change to DST in covered: the time zone indicator changes from +05:00 to +04:00.

ALTERNATIVES?

SAS also has PROC Format to generated one's own format, and there one can use 'directives' to select specific elements of datetime values in a picture format. These directives are locale sensitive, so that specifying the name of the month (%b for an abbreviation, %B for the full month name) will generate the name for the locale that is in effect when the format is being used.

But these directives are not able to adjust UTC values to the local time zone. And an additional problem is that these directives are not supported in the SG-procedures (SG stands for Statistical Graphics since these procedures were originally introduced for the procedures in the SAS/Stat®, but are now part of SAS Base product, and offer more advanced options then the procedures in SAS/Graph®). When using a picture format with a directive like %b, the result shows just that: "%B", and not the intended name of the month.

What we in fact want is a format where we can select which elements of a datetime value have to be shown, how they should be formatted, adjusted for the time difference between the local time and UTC, if needed accounting for daylight savings time.

By combining two lesser known possibilities of SAS this can be accomplished:

- Defining new functions with PROC FMCP.
- Using functions inside a format.

CREATING A FUNCTION WITH PROC FMCP

SAS offers PROC FMCP, which is short for the SAS Function Compiler. It enables you to define new functions that can be used like any existing function in SAS: taking parameters, executing code on those parameters, and returning a result. (One can also define subroutines that can be called using the CALL syntax, but that will not be described here.)

The first function we will describe here takes three parameters.

- A UTC datetime value.
- The difference in hours between the local time and UTC, regardless of DST. This difference is usually an integer, but not necessarily so.
- A three character value indicating whether the DST rules for the USA or for the European Union ('EUR') has to be followed. Any other value then 'USA' or 'EUR' will mean that no adjustment for DST will be performed.

This first function we define will return the adjusted datetime value.

The PROC statement must indicate in which SAS data set the definitions have to stored. Within the data set the definitions are organized in 'packages', so this becomes a three level definition.

```
proc fcmp outlib=work.functions.utc;
```

Next the function is introduced: name and parameters.

```
function utc2local (UTCdt , TimeZone, DST $ ) ;
```

The $ signs indicates that the third parameter is a character value.

Now initialize the difference to be added for DST to zero, and determine the current year.

Then check which DST rules to follow: USA, EUR or none.

```
DSTdiff=0 ;
year = year ( datepart ( UTCdt ) ) ;
select ( DST ) ;
```

Then for the USA determine the dates that DST starts and end: the second Sunday in March, and the first Sunday in November. (The possibilities of the function NWKDOM were learned from Rick Wicklin in his SAS blog "the DO loop".)

Convert the two dates then to datetime values by setting the time at 2:00.

```
      when ( "USA" ) do ;
            dst_beg = nwkdom ( 2 , 1 ,  3 , year ) ;
            dst_end = nwkdom ( 1 , 1 , 11 , year ) ;
            dst_beg = dhms ( DST_beg , 2 , 0 , 0 ) ;
            dst_end = dhms ( DST_end , 2 , 0 , 0 ) ;
```

Now subtract the time zone difference. It was given in hours, so multiply by 3600 to get the number of seconds. It has to be subtracted, because if you are in a time zone that has a +6 difference with UTC, i.e. west of Greenwich, your local time is 6 hours *behind*.

Then check if the local time is between the beginning and end datetime values for DST, and if so, set DSTdiff to one hour.

```
            LOCdt = UTCdt - 3600 * timeZone ;
            if LOCdt >= DST_beg and UTCdt < DST_end then DSTdiff=3600;
      end ;
```

For Europe the logic is different.
We need the last Sundays of the March and October, and this can be accomplished by asking for the *fifth* occurrence in that month. If there is no fifth Sunday, the function will return the last one (i.e., the fourth). The time is set to 1:00.

And instead of checking the adjusted local value, the original UTC datetime is checked against the beginning and end of DST.

```
      when ( "EUR" ) do ;
            dst_beg = nwkdom ( 5 , 1 , 3  , year ) ;
            dst_end = nwkdom ( 5 , 1 , 10 , year )
            dst_beg = dhms ( DST_beg , 1 , 0 , 0 ) ;
            dst_end = dhms ( DST_end , 1 , 0 , 0 ) ;
            LOCdt = UTCdt - 3600 * timeZone ;
            if UTCdt >= DST_beg and UTCdt < DST_end then DSTdiff=3600;
      end ;
      otherwise LOCdt = UTCdt - 3600 * timeZone ;
  end ;
```

Adjust the value for DST (DSTdiff then will be zero or set to 3600), and specify the value as the return value, and finish the definition.

```
LOCdt = LOCdt + DSTdiff ;
return ( LocDT ) ;
endsub;
run ;
```

The function in this form returns a new datetime, and now can be used in a DATA step. The only thing needed is to tell SAS that a function is available in the package we selected when initializing PROC FMCP.

```
options cmplib=(work.functions);
data;
utcdt = '1apr2020:11:00'dt ;
AmsterdamTime = utc2local ( utcdt , -1 , 'EUR' ) ;
run ;
```

The purpose was to have a function that can be used within a format. For that it has to produce a character string, and it can take only one parameter. That parameter is necessarily the UTC time that has to be adjusted, so there is no room for the time zone or for the method to determine start and end of DST. This means that for each combination a separate function has to be specified. The specific elements from the datetime value that have to be returned in the character string can vary, depending on requirements.

In the following example only the time is returned, and an indicator is appended to the string if the local time was adjusted for DST.

The first line of the function definition now are as follows. The same SAS data set and package is used to store the function. The function is called "utc2m1EUR" (UTC to time in time zone 'minus 1', with European rules).

It has only one numeric parameter, and the result will be character (hence the $ at the end).

The variable LOCstring will hold the result to return at the end.

And the variables that specify the time zone value and the DST rule is fixed for this specific function.

```
proc fcmp outlib=work.functions.utc;
function utc2m1EUR (UTCdt  ) $ ;
length LOCstring $ 40 ;
DST= 'EUR' ;
timezone=-1;
```

Now exactly the same code as in the previous function can be used. This could have been done by putting that in a separate function, and nesting that, but we have kept it simple.

At the end of the function the adjusted LOCdt value has to be translated to the string that one wants to see.

```
if DSTdiff > 0 then do ;
    LOCstring = cats ( LOCstring , '^{unicode 2609}' ) ;
end ;
```

The indicator to mark the values that have been subject to DST adjustment is the astronomical sign for the sun: ☉. This is done by using the ODS syntax to insert characters from the Unicode table: the ODS escape character (here: ^), followed by the Unicode code point.
In the appendix a generalized version of the code is given, using macro syntax, in order to define a set of functions for different time zones and DST rules.

CREATING FORMATS WITH A FUNCTION

PROC FMCP is relatively unknown, and the possibility to use a function, whether it is an existing SAS function, or one created with this procedure, in a value format is almost equally unknown. A value format gives for a specified range of the value to be formatted usually the string to display instead of that format. The documentation explains that that string also can specify an existing format by giving that between square brackets. A format for a date value thus could define the following ranges:

```
low -< '1jan1900'd = 'Before 1900'
'1jan1900'd - high = [year4.]
```

This will display any date before (and not including) January 1st, 1900, as the string 'Before 1900', and for all later dates will show the year.

The documentation also gives an example using this, and from a remark there it follows that if the characters between the brackets are not followed by a point, but by opening and closing parentheses, this will call the *function* with that name. It implies that the value to be formatted will be passed on as the parameter for the function.

This enables us to use the kind of function we have defined to use in a format. The format is defined here with the same name as the function, but this is by no means necessary.

```
proc format ;
value UTC2m1EUR (default=30)
    low-high = [utc2m1EUR()]
;
run ;
```

And this makes it possible to store all date time values in UTC form, use it in UTC form through any manipulations, and when creating output translate it 'on the fly' into the local datetime value, using DST adjustments if needed, and selecting the date and time elements we need. The following table shows the result, using the format and function described.

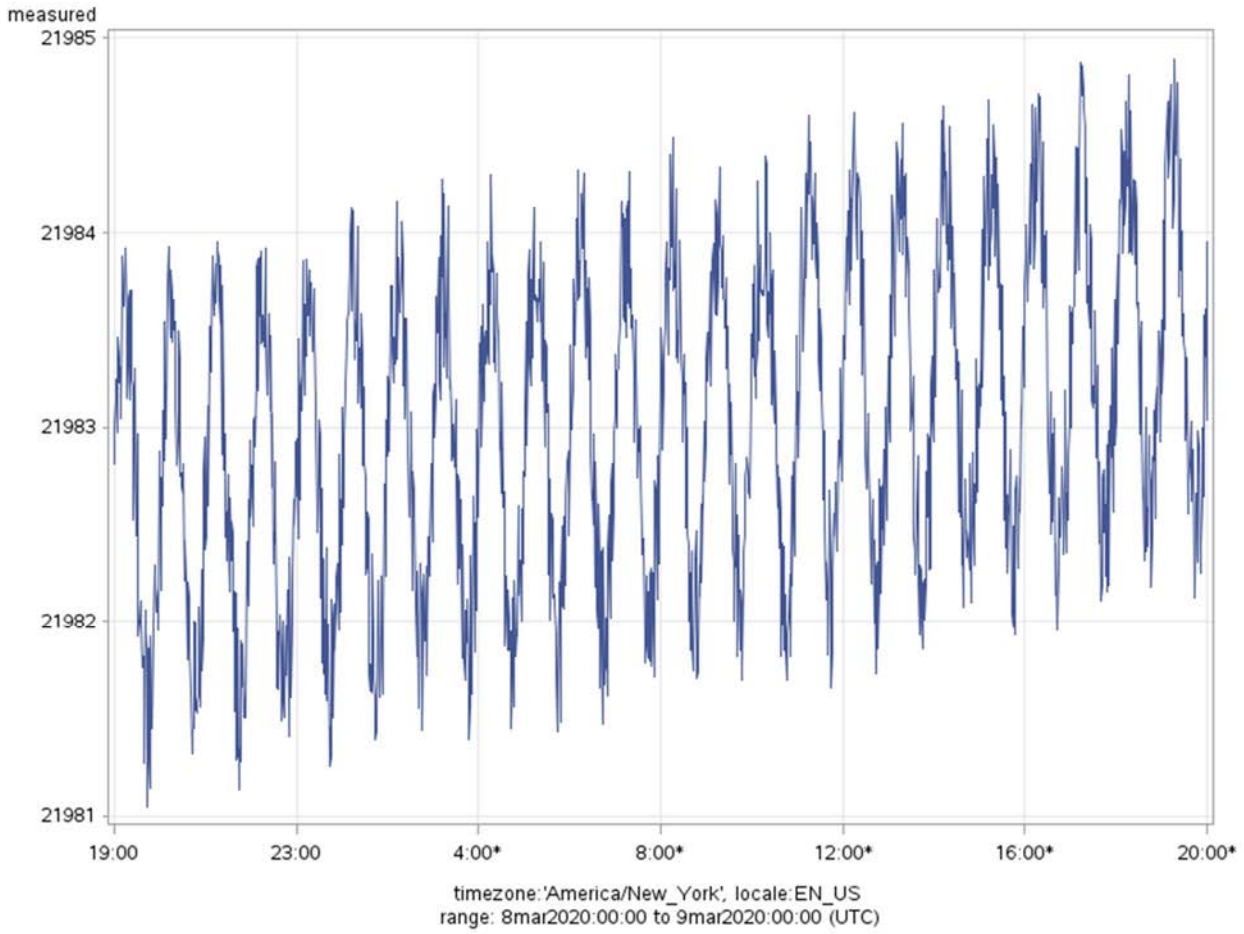
timezone:'Europe/Amsterdam', locale:NL_NL

UTC, NL format	ISO 8601 DX format	user function EUR
25 oktober 2020 00:57:00 uur	20201025T015700+0100	2:57⊙
25 oktober 2020 00:58:00 uur	20201025T015800+0100	2:58⊙
25 oktober 2020 00:59:00 uur	20201025T015900+0100	2:59⊙
25 oktober 2020 01:00:00 uur	20201025T020000+0100	2:00
25 oktober 2020 01:01:00 uur	20201025T020100+0100	2:01
25 oktober 2020 01:02:00 uur	20201025T020200+0100	2:02
25 oktober 2020 01:03:00 uur	20201025T020300+0100	2:03

Display 3. Table showing different formats

This table was produced with SAS 9.4, M3. Note that while the user generated function switches from DST to normal time at UTC 01:00 h, the SAS ISO format had done so earlier, which is not correct. SAS 9.4 M6 will produce a correct result.

This format also can be used in a plot, generating a result that leaves much more room for the graph itself, because the tick marks are now limited to the essential information.
In order not to complicate things here to try and get the astronomical symbol ⊙ in the graph, this has been replaced in the format by a simple *, which starts to appear at the 3rd tick mark, when DST has started in the time zone that currently is in effect.



Display 4. Graph using user format with user function

OPTIONS FOR FURTHER DEVELOPMENT

The latest maintenance releases for SAS9.4 have additional functions, that make it possible to query the current time zone settings, and the attributes of the time zone. That will make it possible to generalize the functions and formats described here, to a next level. Up till now it is necessary to create a separate function *and* format for *each* time zone and for *each* DST rule. That is inevitable because when used inside the format the function cannot take a second or third parameter.

Now the function can be enhanced to include logic that determines the time zone and the time difference with respect to UTC.

That does not yet specify the DST rules that should apply. Whether to determine that from the time zone or the locale setting is a matter of choice. This choice will also depend on the question how complete that logic has to be, because in the real world the variation is large. Countries or region sometimes change the logic over time, and there are exceptions (all states in the 'contiguous' USA observe DST, except Arizona; but the Navajo Nation, partly in Arizona, *does* observe DST; while the Hopi Nation, lying within the Navajo Nation, follows the rest or Arizona).

CONCLUSIONS

The flexibility of an application is greatly improved by having the logic to determine when, where and how a difference in time zone and possible daylight saving time has to be applied inside the code for a function. It also provides the possibility to determine exactly how the result should be displayed.

Any change in local habits or rules can easily be incorporated. This might pertain to the question if DST is used, when it starts and end, and also can implement differences over time.

PROC FMCP is very useful tool to implement this functionality, especially in combination with the option to use such functions within a SAS format.

REFERENCES

Verne, Jules. 1872. *Le tour du monde en quatre-vingts jours.*
Available at the Gutenburg Project, no 800
http://www.gutenberg.org/ebooks/800

Id. *Around the World in Eighty Days* (English translation)
Available at the Gutenberg Project, no 103
http://www.gutenberg.org/ebooks/103

SAS® 9.4 National Language Support (NLS): Reference Guide, Fifth Edition
At https://go.documentation.sas.com/
Last accessed 12 February 2020

Wicklin, Rick. 2012. *Computing the onset and end of daylight saving time in SAS: The EASY way!*
At http://blog.sas.com/
Last accessed 27 February 2020

CONTACT INFORMATION

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

Frank Poppe
PW Consulting (the Netherlands)
+31 6 2264 0854
Frank.Poppe@PWconsulting.nl
http://www.pwconsulting.nl

APPENDIX

```
/* Create example data */

data test (keep = UTCdt y) ;
pi = constant ('pi') ;
step = 60 ;
do UTCdt = '1jan2020:00:00'dt to '1jan2021:00:00'dt by step ;
      t = timepart ( UTCdt ) ;
      d = datepart ( UTCdt ) ;
      rad = 2 * pi * t / 3600 ;
      fuzz = ranuni ( 123454321 ) ;
      y = d + fuzz + t/86400 + sin ( rad )  ;
      output ;
end ;
run ;

/* Create awkward examples */

%let DTstart = 8mar2020:00:00 ;
%let DTend = 9mar2020:00:00 ;
options timezone= 'America/New_York' locale = 'EN' ;
footnote ;

ods html
      path = "&path"
      file = "awkward graph.html"
;

symbol1 interpol=join value = none;
axis1
      minor = none
      order = "&DTstart"dt to "&DTend"dt by dthour4
      label = (
            j = c "timezone:%sysfunc(getoption(timezone)),
locale:%sysfunc(getoption(locale)) "
            j = c "range: &DTstart to &DTend (UTC)"
      )
      value = ( height = 10pt )
;
axis2
      minor = none
      label = ( 'measured' )
      value = ( height = 10pt )
;
title;

proc gplot data = test (where = ( utcdt between "&DTstart"dt and "&DTend"dt )
)  ;
plot y * utcdt
      /     haxis=axis1 vaxis=axis2 autovref autohref
;
format utcdt b8601dx. ;
run ;
quit ;

ods _ALL_ close ;
```

```
options timezone= 'Europe/Amsterdam' locale = 'NL' ;

ods html
      path = "&path"
      file = "awkward table.html"
;
title "timezone:%sysfunc(getoption(timezone)),
locale:%sysfunc(getoption(locale)) " ;
ods escapechar '' ;
proc report data = test
      ( where = ( utcdt between "&DTstart"dt and "&DTend"dt  ) )
;
columns utcdt ( 'ISO8601 DX' isofmtB isofmtE ) ( 'timepart' timefmt
timefmtNL)  ;
define utcdt / 'UTC, NL format' display format = nldatm. ;
define isofmtB  / 'Basic' computed format = b8601dx. ;
define isofmtE  / 'Extended' computed format = e8601dx. ;
define timefmt / 'standard' computed format = tod. ;
define timefmtNL / 'NL format' computed format = nldatmtm. ;
compute isofmtB ;
      isofmtB = utcdt ;
endComp ;
compute isofmtE ;
      isofmtE = utcdt ;
endComp ;
compute timefmt ;
      timefmt=utcdt ;
endComp ;
compute timefmtNL ;
      timefmtNL=utcdt ;
endComp ;

run ;
ods _ALL_ close ;
```

```
/* Generalized code to create a set of functions and formats,
For several time zones in Europe and the USA
*/


%macro UTC2EUR ;
%do zone = -2 %to 0 ;

%if &zone < 0 %then %let pm = M ;
%else %let pm = P ;
%let hour = %sysfunc ( abs ( &zone ) ) ;

function utc2&pm.&hour.EUR (UTCdt  ) $ ;
length locstring $ 40 ;

DST= 'EUR' ;
timezone=&zone ;
year = year ( datepart ( utcdt ) ) ;
DSTdiff = 0 ;
checkDST = upcase ( DST ) ;
select ( checkDST ) ;
      when ( "USA" ) do ;
            dst_beg = nwkdom ( 2 , 1 ,  3 , year ) ; /* 2nd Sunday in March
*/
            dst_end = nwkdom ( 1 , 1 , 11 , year ) ; /* first Sunday in
November */
            dst_beg = dhms ( DST_beg , 2 , 0 , 0 ) ;
            dst_end = dhms ( DST_end , 2 , 0 , 0 ) ;
            LOCdt = UTCdt - 3600 * timeZone ;
            if LOCdt >= DST_beg and UTCdt < DST_end then DSTdiff=3600;
      end ;
      when ( "EUR" ) do ;
            dst_beg = nwkdom ( 5 , 1 , 3  , year ) ; /* last Sunday of March
*/
            dst_end = nwkdom ( 5 , 1 , 10 , year ) ; /* last Sunday of
October */

            dst_beg = dhms ( DST_beg , 1 , 0 , 0 ) ;
            dst_end = dhms ( DST_end , 1 , 0 , 0 ) ;

            LOCdt = UTCdt - 3600 * timeZone ;
            if UTCdt >= DST_beg and UTCdt < DST_end then DSTdiff=3600;

      end ;
      otherwise LOCdt = UTCdt - 3600 * timeZone ;
end ;

LOCdt = LOCdt + DSTdiff ;

LOCstring = put ( timepart (locDT) , time5. ) ;
/*if DSTdiff then LOCstring = cats ( LOCstring , "^{unicode 2609}" ) ;*/
if DSTdiff then LOCstring = cats ( LOCstring , "*" ) ;

return ( LocString ) ;

endsub;
%end ;
%mend ;
```

```
%macro UTC2USA ;
%do zone = 5 %to 8 ;

%if &zone < 0 %then %let pm = M ;
%else %let pm = P ;
%let hour = %sysfunc ( abs ( &zone ) ) ;

function utc2&pm.&hour.USA (UTCdt  ) $ ;
length locstring $ 40 ;

DST= 'USA' ;
timezone=&zone ;
year = year ( datepart ( utcdt ) ) ;
DSTdiff = 0 ;
checkDST = upcase ( DST ) ;
select ( checkDST ) ;
      when ( "USA" ) do ;
            dst_beg = nwkdom ( 2 , 1 ,  3 , year ) ; /* 2nd Sunday in March
*/
            dst_end = nwkdom ( 1 , 1 , 11 , year ) ; /* first Sunday in
November */
            dst_beg = dhms ( DST_beg , 2 , 0 , 0 ) ;
            dst_end = dhms ( DST_end , 2 , 0 , 0 ) ;
            LOCdt = UTCdt - 3600 * timeZone ;
            if LOCdt >= DST_beg and UTCdt < DST_end then DSTdiff=3600;
      end ;
      when ( "EUR" ) do ;
            dst_beg = nwkdom ( 5 , 1 , 3  , year ) ; /* last Sunday of March
*/
            dst_end = nwkdom ( 5 , 1 , 10 , year ) ; /* last Sunday of
October */

            dst_beg = dhms ( DST_beg , 1 , 0 , 0 ) ;
            dst_end = dhms ( DST_end , 1 , 0 , 0 ) ;

            LOCdt = UTCdt - 3600 * timeZone ;
            if UTCdt >= DST_beg and UTCdt < DST_end then DSTdiff=3600;

      end ;
      otherwise LOCdt = UTCdt - 3600 * timeZone ;
end ;

LOCdt = LOCdt + DSTdiff ;

LOCstring = put ( timepart (locDT) , time5. ) ;
/*if DSTdiff then LOCstring = cats ( LOCstring , "^{unicode 2609}" ) ;*/
if DSTdiff then LOCstring = cats ( LOCstring , "*" ) ;

return ( LocString ) ;

endsub;
%end ;
%mend ;

proc fcmp outlib=work.functions.utc;
```

```
%UTC2EUR ;
%UTC2USA ;
run ;
quit ;

options cmplib=(work.functions);

%macro FMTs ;
%do zone = -2 %to 0 ;

%if &zone < 0 %then %let pm = M ;
%else %let pm = P ;
%let hour = %sysfunc ( abs ( &zone ) ) ;
value UTC2&pm.&hour.EUR (default=30)
      low-high = [utc2&pm.&hour.EUR()]
;
%end ;

%do zone = 5 %to 8 ;

%if &zone < 0 %then %let pm = M ;
%else %let pm = P ;
%let hour = %sysfunc ( abs ( &zone ) ) ;
value UTC2&pm.&hour.USA (default=30)
      low-high = [utc2&pm.&hour.USA()]
;
%end ;

%mend ;

proc format ;
%fmts ;
run ;
```

```
/* Create nicer looking examples */

options timezone= 'Europe/Amsterdam' locale = 'NL' ;
title "timezone:%sysfunc(getoption(timezone)),
locale:%sysfunc(getoption(locale)) " ;
ods _ALL_ close ;
ods html
      path = "&path"
      file = "better table.html"
;
ods escapechar '^' ;
proc report data = test
      ( where = ( utcdt between '25oct2020:00:57'dt and '25oct2020:02:00'dt )
)
;
columns utcdt isofmt userFunc ;
define utcdt / 'UTC, NL format' display format = nldatm. ;
define isofmt  / 'ISO 8601 DX format' computed format = b8601dx. ;
define userFunc / 'user function  EUR' computed format = utc2m1EUR. ;
compute isofmt ;
      isofmt = utcdt ;
endComp ;
compute userFunc ;
      userFunc=utcdt ;
endComp ;

run ;
ods _ALL_ close ;

%let DTstart = 8mar2020:00:00 ;
%let DTend = 9mar2020:00:00 ;
options timezone= 'America/New_York' locale = 'EN' ;
footnote ;
ods html
      path = "&path"
      file = "improved graph.html"
;
symbol1 interpol=join value = none;
axis1
      minor = none
      order = "&DTstart"dt to "&DTend"dt by dthour4
      label = (
            j = c "timezone:%sysfunc(getoption(timezone)),
locale:%sysfunc(getoption(locale)) "
            j = c "range: &DTstart to &DTend (UTC)"
      )
      value = ( height = 10pt )
;
axis2
      minor = none
      label = ( 'measured' )
      value = ( height = 10pt )
;
title;
proc gplot data = test (where = ( utcdt between "&DTstart"dt and "&DTend"dt )
)  ;
plot y * utcdt
      /     haxis=axis1 vaxis=axis2 autovref autohref
```

```
;
format utcdt utc2p5USA. ;
run ;
quit ;

ods _ALL_ close ;
```