SAS® GLOBAL FORUM 2020 #SASGF

Paper 4801-2020

Time Is on Your Side. Yes It Is. Using Base SAS® to Manipulate Time

Carole Jesse, Idaho National Laboratory

ABSTRACT

Time can be tricky. In relational database management systems (RDBMS) there can be many time-based (temporal) components (for example, datetime, date, and time fields). They are typically stored as numeric data types in the native RDBMS format for temporal components. Sometimes they are stored as character fields in the RDBMS. The way that the SAS® software interprets and reads temporal components depends on several things. The challenge of reading and converting them into useful variables for data analysis in SAS requires some finesse, but any SAS programmer or analyst can master this with a few tips and tricks. This presentation helps you understand how temporal components are stored and formatted in RDBMS and subsequently how to take advantage of all that Base SAS® has to offer for working with temporal components. The use of SAS INFORMATs, FORMATs, automatic macro variables (like &SYSDATE9), and useful SAS functions (like INTCK and/or INTNX) are demonstrated to inspire programmers and analysts to tackle any temporal component manipulation needed to render useful analyses of their data.

INTRODUCTION

Time components are everywhere in RDBMS and they are frequently critical to answering business questions. Often the focus of these questions relates to understanding a business process and identify gaps, bottlenecks, and sweet spots. Some examples include:

- Information Technology: How many views has this web page had today? Last week? Is there a peak 'day of week' or 'time of day' for page views?
- Procurement: What is the distribution of the 'time between' Order Submission and Order Approval in the procurement system? From Order Approval to Order Placement? From Order Placement to Order Delivery?
- Talent Acquisition: What is the average time to fill a position, from posting the position to offer acceptance?
- Marketing: How long did it take for existing customers to respond to the current promotional offer? Did the time to respond depend on the marketing channel: postal mail versus marketing email?

As a programmer or analyst, it is inevitable that you will need to tackle the concepts of time in your data. While there are some very powerful SAS tools for handling time manipulation problems, SAS/ETS® software for example, you may not have that product in your licensing. However, Base SAS has many features that allow you to perform some of the basic manipulations that are required. These techniques should be in your wheelhouse for prepping data using Base SAS alone.

This paper will appeal most to those who want to expand their Base SAS programming skills. Those at beginner and intermediate skills levels will benefit, especially if they have not dealt much with the topic of time-based (temporal) components before. It is assumed that the reader knows the basics of SAS programming such as writing LIBNAME statements to create a *libref*, constructing simple SAS DATA steps and/or SAS SQL procedure programs. Additionally, it is assumed that the reader knows how to connect to their RDBMS and how to reference a schema with a *libref*. Those basic concepts are not covered here, however, there are many resources available on the web that do, one is included in RECOMMENDED READING.

This paper introduces the reader to the basics of temporal components in SAS and tackles both simple and complex manipulations of them, including, in roughly this order:

1. SAS literals, INFORMATs and FORMATs for the date, time and datetime data types
2. Transformation of temporal components that are stored natively in the RDMBS as character and must be converted properly in SAS
3. Transformation of one temporal component to another: such as combining date and time into datetime
4. Calculation of durations, or 'time between events'
5. Some of the SAS functions utilized include INPUT, INTCK, DHMS, INTNX

For most of this paper publicly available data is utilized from the SAS Global Forum 2020 (SASGF2020) website. This allows the reader to replicate the concepts and techniques demonstrated. While these data are delivered as a Microsoft® Excel® file, all the concepts explored with these data are applicable to working with RDBMS. The final section of the paper demonstrates putting it all together against an Oracle® RDBMS and incorporating some automation.

Content generation for this paper utilizes SAS version 9.4 M3 with SAS Enterprise Guide® version 7.1 as the interface. Additionally, the SAS product used for connection to both Oracle and Microsoft SQL Server® RDBMS is SAS/ACCESS® to ODBC. There may be differences between this environment and what readers have available to them.

The full SAS programs are contained in the APPENDIX and at the Author's GitHub (see CONTACT INFORMATION section). Key portions are highlighted in the sections of the paper.

TIME IS A DIFFERENT SORT OF NUMERIC

While time is a continuous numeric, it is broken into discretized units: year, month, day, hour, minute, and second. This seems obvious, but this is unlike any other continuous numeric you can think of. In addition, the concept of the date/time continuum captures:

Events (a single anchored moment in time), such as "the presentation of SASGF2020 paper #4801 is on April 1, 2020 at 10:00 AM."

Durations (a window of time without regard to beginning/end), such as "SASGF2020 paper #4801 is presented as a 30-minute Breakout Session."

The concepts of events and durations drove the temporal terminology used for databases and the original data types referred to as TIMESTAMPs and INTERVALs.

SAS on the other hand treats temporal components as DATE and TIME, from which DATETIME comes naturally as a combination of the two, a slightly different lexicon from the database world. It is important to note that whether you are dealing with an event or a duration is determined by the context of how you are using the data.

TEMPORAL COMPONENTS IN SAS

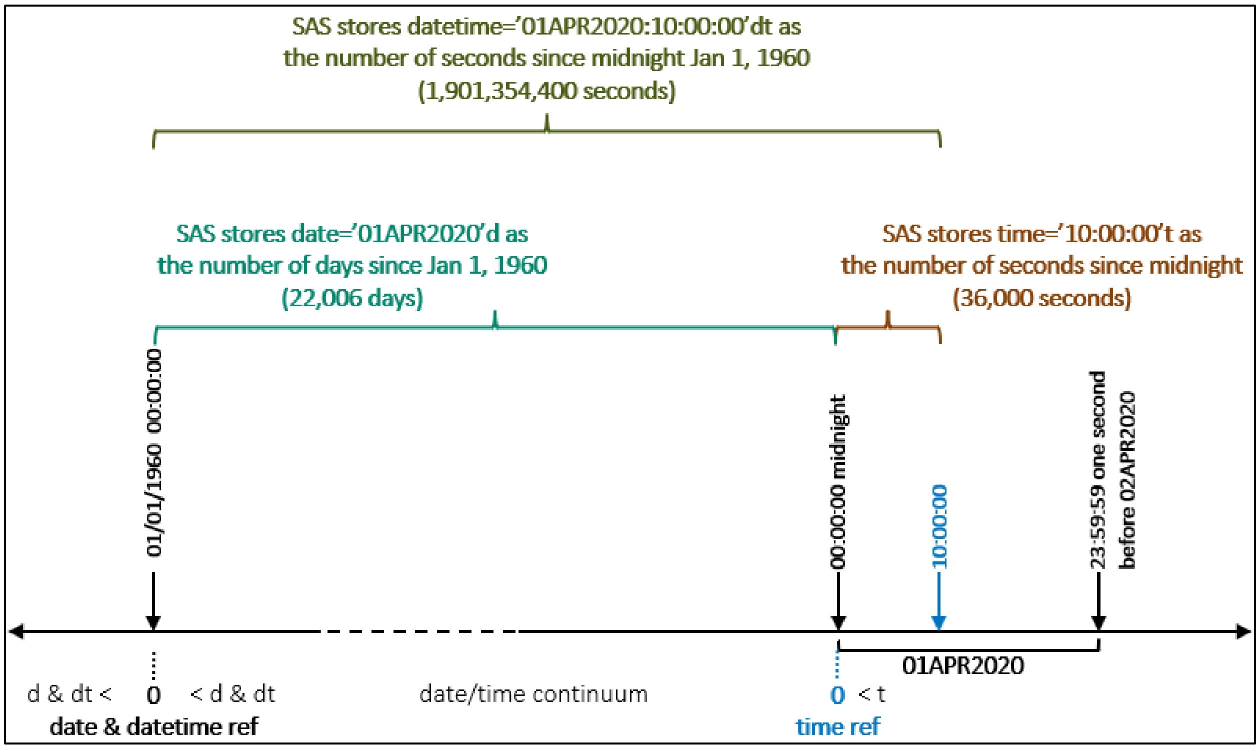
SAS stores temporal components as integer counters from a reference point in time. The units of the counter depend on whether you are dealing with a date (unit is number of days), or a time/datetime (unit for either is number of seconds). Furthermore, for both date and datetime, the counters can be positive (forward in time from the reference point) or negative (backward in time from the reference point). If you are dealing with a time temporal component, the counter is always positive since it is relative to Midnight within a day.

The SAS DATE component is a measure of the number of days since the reference point January 1, 1960.

The SAS TIME component is a measure of the number of seconds since the reference point of Midnight (hours/minutes/seconds or hh:mm:ss = 00:00:00, where this notation assumes a 24-hour clock, i.e. 10 PM is 22:00:00).

The SAS DATETIME component is a measure of the number of seconds since the reference point of Midnight, January 1, 1960.

To help visualize these concepts, consider the date and time for the start of the presentation for SFG2020 paper #4801: April 1, 2020, at 10:00 AM. Figure 1 depicts the relationships between the date/time continuum, the presentation event, and how SAS stores the date, time, and datetime temporal components.



SAS stores datetime='01APR2020:10:00:00'dt as the number of seconds since midnight Jan 1, 1960 (1,901,354,400 seconds)
SAS stores date='01APR2020'd as the number of days since Jan 1, 1960 (22,006 days)
SAS stores time='10:00:00't as the number of seconds since midnight (36,000 seconds)
01/01/1960 00:00:00
00:00:00 midnight
10:00:00
23:59:59 one second before 02APR2020
01APR2020
d & dt < 0 < d & dt
date & datetime ref
date/time continuum
0 < t
time ref

Figure 1 SASGF2020 Paper #4801 Presentation Time in SAS

SAS LITERALS AND FORMATS FOR DATE, TIME AND DATETIME

The APPENDIX SAS program for this section is 1_SASliterals.sas.

Since the measures of date, time and datetime are all numeric counters based on a reference point, interpretation of the counters is not intuitive to the human mind; we just aren't taught to think about it this way. For ease of interpretation, it is crucial to understand how to reference these components with their respective SAS literals and how to apply appropriate SAS FORMATs to render them interpretable.

While SAS stores time, date and datetime as 'counters from a reference point' you can apply SAS FORMATs to them for easy interpretation. Table 1 shows the SAS literal syntax for each of the three temporal components, with specific values as they relate to the date and time of the SASGF2020 paper #4801 presentation.

SAS data type	SAS literal syntax	Logical use in DATA step	Results
date	`'ddMONyyyy'd` the d indicates this is not a quoted character string, but rather a date	`if SGF_DATE < '01APR2020'd then output;`	a subset of SASGF2020 catalog data: sessions prior to April 1
time	`'hh:mm:ss't` the t indicates this is not a quoted character string, but rather a time	`if SGF_TIME_beg = '10:00:00't then output;`	a subset the SASGF2020 catalog data: sessions starting at 10:00 AM, irrespective of day
datetime	`'ddMONyyyy:hh:mm:ss'dt` the dt indicates this is not a quoted character string but rather a datetime	`if SGF_DATETIME >= '01APR2020:10:00:00'dt then output;`	a subset of SASGF2020 catalog data: sessions at 10:00 AM or later on April 1

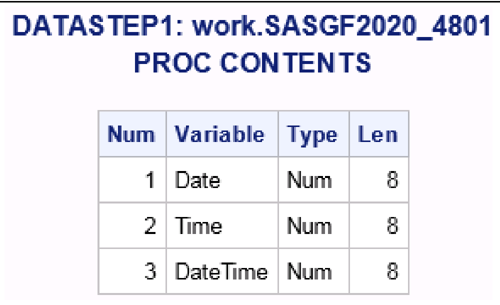
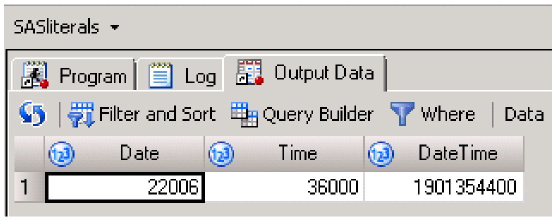
Table 1 SAS Literals for Date, Time, and Datetime for April 1, 2020, 10:00 AM

For learning purposes, you can run simple SAS DATA steps to see the difference between the unformatted and formatted SAS literals as they relate to the created SAS variables SGF_DATE, SGF_TIME_beg and SGF_DATETIME.

To better illustrate this, refer to Table 2. Column 1 of Table 2 shows the two SAS DATA steps from the SAS program 1_SASliterals. The two DATA steps differ only by the inclusion of the SAS FORMAT statement. Column 2 of Table 2 reveals the results of a subsequent CONTENTS procedure and provides a look at how the data set is rendered in SAS Enterprise Guide.

With the inclusion of the FORMAT statement in the second DATA step, notice the appearance of the applied FORMATs in the PROC CONTENTS result, as well as the change in SAS Enterprise Guide icons from standard numeric to actual calendar/time icons. The formats chosen in the FORMAT statement are personal preference, for demonstrative purposes, but any legitimate SAS FORMAT can be utilized.

Source Code	PROC CONTENTS and SAS/EG Rendering

`/* DATA STEP 1 */`
 	
`data work.SASGF2020_4801;`
`   Date='01APR2020'd;`
`   Time='10:00:00't;`
`   DateTime='01APR2020:10:00:00'dt;`
`run;`
	 SAS/EG Rendering 

DATASTEP1: work.SASGF2020_4801
PROC CONTENTS

Num	Variable	Type	Len
1	Date	Num	8
2	Time	Num	8
3	DateTime	Num	8

SAS/EG Rendering

	Date	Time	DateTime
1	22006	36000	1901354400

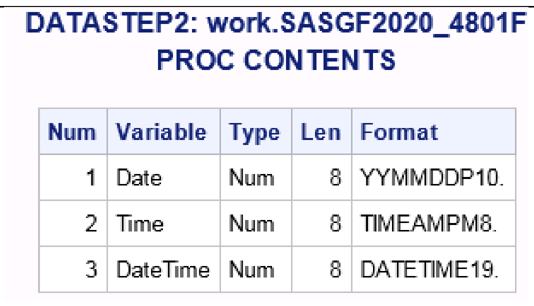
Source Code

```
/* DATA STEP 2 */

Data work.SASGF2020_4801F;
   Date='01APR2020'd;
   Time='10:00:00't;
   DateTime='01APR2020:10:00:00'dt;


format
Date YYMMDDp10.
Time TIMEAMPM8.
DateTime DATETIME19.;
run;
```

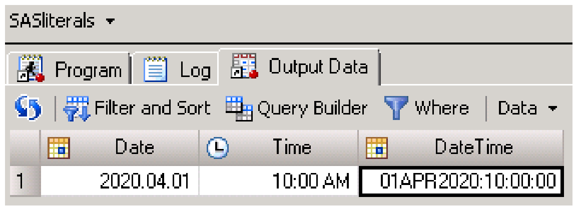
PROC CONTENTS and SAS/EG Rendering



DATASTEP2: work.SASGF2020_4801F
PROC CONTENTS

Num	Variable	Type	Len	Format
1	Date	Num	8	YYMMDDP10.
2	Time	Num	8	TIMEAMPM8.
3	DateTime	Num	8	DATETIME19.

SAS/EG Rendering



	Date	Time	DateTime
1	2020.04.01	10:00 AM	01APR2020:10:00:00

Table 2 SAS DATA Steps to Compare Unformatted/Formatted SAS Literals

IMPORT THE SASGF2020 CATALOG

The APPENDIX SAS program for this section is 2_ReadSAScatalog.sas, which also includes instructions for finding the data at the SASGF2020 website.

Now that you have a good grasp of working with the three SAS literals and their associated FORMATs, it's time to work through some of the typical issues that arise in dealing with temporal component variables brought in from data sources like Excel, RDBMS, and even existing SAS data sets.

The SAS program 2_ReadSAScatalog starts with the IMPORT procedure to read the SASGF2020 Catalog Excel file into SAS, then PROC SQL performs some column selection and renaming to create the SAS data set work.SAScatalog. Once the data are in SAS a quick PROC CONTENTS reveals some issues with these data. Specifically, there are three variables that are clearly temporal but stored as character on the SAS side after the import. One reason for this may be that the cell formats are set to 'General' in the Excel spreadsheet, rather than actual Excel 'Date' and 'Time' cell formats. However, going into the Excel file and making those changes prior to PROC IMPORT is the wrong way to go about the correction. Rather, the conversion to true numeric date and time data types can happen after the import through a PROC SQL. This approach is identical to the case when the data source is a RDBMS and the data are stored natively as character there. Unless you are the Database Administrator, the change from character to numeric can't happen there. But since you are a SAS Programmer/Analyst you make the conversion through a PROC SQL!

The SASGF2020 Catalog Excel file does not provide the duration of sessions, but since StartTime and EndTime are included, you can calculate session durations using their true numeric versions (SGF_TIME_beg and SGF_TIME_end, respectively). SASGF2020 Breakout Sessions are either 30- or 60-minutes long and allow a 10-minute Q&A at the end. Thus, the duration calculations in the PROC SQL include 'with' (_wQA) and 'without' (_woQA) the Q&A window.

For the sake of the learning experience, the PROC SQL in program 2_ReadSAScatalog demonstrates how to create a true numeric datetime variable for the start of each session.

Table 3 depicts the existing layout for the three variables after running 2_ReadSAScatalog, and the desired transformations and calculations. Fixing the temporal component issues (the conversions) and creating new temporal components (the calculations) are the subject of the next section and the associated SAS program.

work.SAScatalog Layout			work.SAScatalog Desired Additions		
VarName	VarFormat	SampleValue	VarName	VarFormat	SampleValue
Day	$10.	04/01/2020	SGF_DATE	MMDDYYs10.	04/01/2020
StartTime	$8.	10:00 AM	SGF_TIME_beg	TIMEAMPM8.	10:00 AM
EndTime	$8.	10:30 AM	SGF_TIME_end	TIMEAMPM8.	10:30 AM
			SGF_DATETIME	DATETIME21.	01APR2020:10:00:00
			SESSN_DUR_wQA	(numeric calc)	30
			SESSN_DUR_woQA	(numeric calc)	20

Table 3 work.SAScatalog Existing Layout and Desired Additions

MODIFY THE SASGF2020 CATALOG

The APPENDIX SAS program for this section is 3_ConvertSAScatalog.sas.

The goals for modifying the SASGF2020 Catalog are to fix the existing issue with the three temporal components, namely convert them from their character versions to true numeric versions, i.e. create variables SGF_DATE, SGF_TIME_beg and SGF_TIME_end via the SAS function INPUT. Once true numeric SAS temporal components are available, they can be used to perform the 'time between events' math required for the two session duration calculations SESSN_DUR_wQA and SESSN_DUR_woQA. This math is made easy via the SAS function INTCK. Lastly, the SAS function DHMS is employed to demonstrate the creation of a true numeric datetime temporal component called SGF_DATETIME representing the start of each session. While technically SGF_DATETIME is not needed here, DHMS is a handy function to know: what if duration is dependent on times that span two different dates? You need to form two datetime temporal components to compute the duration.

FIX THE TEMPORAL COMPONENT ISSUES

To transform the character variables Day, StartTime, and EndTime into their true numeric SAS temporal components, the SAS function INPUT is used.

General syntax for the INPUT function is INPUT(*source*, *informat*.). The INPUT function returns the value that is produced when SAS converts an expression (the *source* parameter) using the specified SAS INFORMAT (the *informat.* parameter). In other words, you use the function to convert a character *source* to a numeric result. In the case of the variable Day, the *informat*. parameter needs to be an appropriate numeric SAS INFORMAT for dates. The INFORMAT ANYDTDTEw. is used to find a <u>date</u> within a string. Inspecting a sample value for the Day string, '04/01/2020', indicates we must allow a width (w) of 10, or INFORMAT ANYDTDTE10..

Drilling into the SAS program for the conversion of Day to SGF_DATE looks like this:

```
PROC SQL;
  create table work.SASCatalog2 as
  select
   ⋮
  INPUT(Day, ANYDTDTE10.) as SGF_DATE  format=MMDDYYs10. ,

   ⋮
from work.SASCatalog;
```

```
   quit;
```

The INPUT function is applied to the Day variable as the *source* parameter, telling SAS that the INFORMAT is ANYDTDTE10. The INPUT function then handles the Day character string, like '04/01/2020', and converts it to a true numeric date temporal component. The resulting variable SGF_DATE will appear as the number of days since Jan 1, 1960, unless the application of the SAS FORMAT MMDDYYs10. is applied. The SAS FORMAT MMDDYYs10. renders the new numeric variable as MM/DD/YYYY to the eye. The 's' in the INFORMAT adds the slash ('/') delineator. By setting the width (w) to 10 the year will have a 4-digit representation.

In a similar fashion, INPUT is used to transform the character variables StartTime and EndTime into their true numeric time temporal components, SGF_TIME_beg and SGF_TIME_end, respectively. However, since these two variables are time data types, the appropriate SAS INFORMAT (ANYDTTME8.) and FORMAT (TIMEAMPM8.) are applied. The INFORMAT ANYDTTMEw. is used to find a time within a string. Inspecting a sample value for the StartTime string, '10:00 AM', implies we must allow a width (w) of 8, or INFORMAT ANYDTTME8. The SAS FORMAT TIMEAMPM8. was chosen to retain the same look in the numeric version as seen with the character representation, allowing AM/PM rather than a 24-hour clock, where seconds aren't needed.

Drilling into the SAS program for the conversions of StartTime and EndTime looks like this:

```
   PROC SQL;
     create table work.SASCatalog2 as
     select
      ⋮
     INPUT(StartTime, ANYDTTME8.) as SGF_TIME_beg  format=TIMEAMPM8. ,
     INPUT(EndTime, ANYDTTME8.)   as SGF_TIME_end  format=TIMEAMPM8. ,
      ⋮
   from work.SASCatalog;
   quit;
```

CREATE NEW TEMPORAL COMPONENT VARIABLES

In the 'Desired Additions' portion of Table 3, all that's left to do is create SGF_DATETIME and the two duration calculations SESSN_DUR_wQA and SESSN_DIR_woQA. The two SAS functions DHMS and INTCK are used to accomplish this in the SAS program 3_ConvertSAScatalog.

SAS Function DHMS

The SAS function DHMS is designed specifically to create a datetime data type from its component values of date, hour, minute, second.

General syntax for the DHMS function is DHMS(*date*, *hour*, *minute*, *second*). However, this function has a special application whenever a SAS date variable exists along with an associated SAS time variable. In this special application, the syntax is DHMS(*DATEvar*, *0*, *0*, *TIMEvar*).

By feeding a SAS date (variable or literal) into the *date* parameter of the function, a SAS time (variable or literal) into the *second* parameter of the function, and replacing the *hour* and *minute* parameters with zero, triggers this special application of the function. Since DHMS creates a SAS datetime data type, it will render as the number of seconds since Midnight, Jan 1, 1960, unless an appropriate SAS FORMAT is applied. In this case the SAS INFORMAT DATETIME21. is applied.

Drilling into the SAS program for the creation of SGF_DATETIME looks like this:

```
PROC SQL;
  create table work.SASCatalog2 as
  select
   ⋮
  DHMS(INPUT(Day, ANYDTDTE10.), 0, 0, INPUT(StartTime, ANYDTTME8.))
   as SGF_DATETIME      format=DATETIME21. ,

  /* ALTERNATE for SGF_DATETIME creation */
  DHMS(calculated SGF_DATE, 0, 0, calculated SGF_TIME_beg)
   as SGF_DATETIME_alt  format=DATETIME21. ,
   ⋮
from work.SASCatalog;
quit;
```

In the above PROC SQL two forms of the DHMS function are provided: 1) the creation of SGF_DATETIME where the INPUT function is nested within the DHMS function, and 2) the creation of the alternate SGF_DATETIME_alt where the previously calculated variables SGF_DATE and SGF_TIME_beg are called within the DHMS function. Providing the two forms, SGF_DATETIME and SGF_DATETIME_alt, is done purely to demonstrate the flexibility of the programming of functions within PROC SQL.

SAS Function INTCK

Durations, or time between events, are common calculations for both reporting and modeling activities. In the INTRODUCTION several examples were provided focusing on the time between events. Luckily Base SAS provides a very powerful function for this sort of calculation, the INTCK function. This function is so powerful and flexible that the reader is encouraged to peruse the online SAS documentation to fully understand it. Only its simplest form is used here.

In its simplest form, the syntax for the INTCK function is INTCK(*interval*, *start-date*, *end-date*).

The *interval* parameter of the function is the unit used to calculate the duration between the *start-date* and *end-date*. Values for *interval* include 'MINUTE', 'DAY', and 'YEAR', but there are many more options for interval! The *start-date* and *end-date* parameters of the function must be SAS temporal components (literals or variables). For the session duration calculations, determining the number of minutes ('MINUTE' interval) between the session start and end times (SGF_TIME_beg and SGF_TIME_end) is sufficient, at least for Breakout Sessions which run 30-60 minutes.

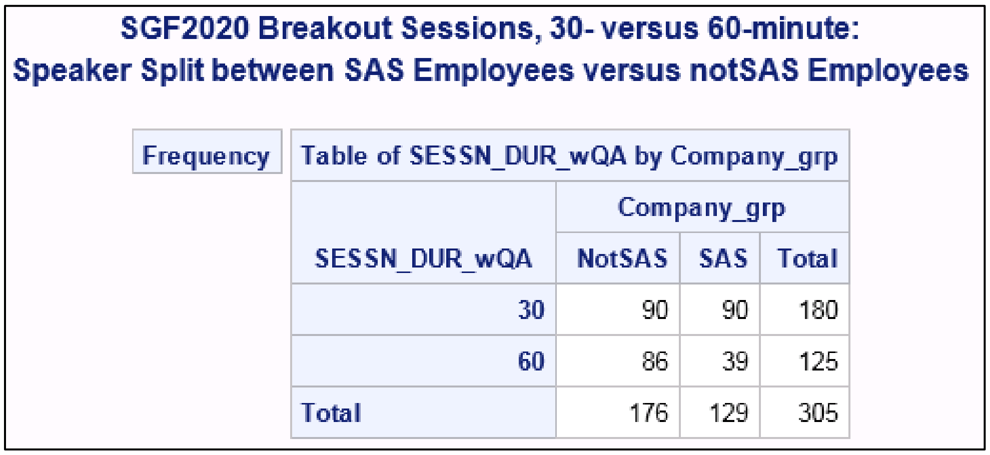
Drilling into the SAS program for the creation of SESSN_DUR_wQA and SESSN_DUR_woQA looks like this:

```
PROC SQL;
  create table work.SASCatalog2 as
  select
   ⋮
  INTCK('MINUTE', calculated SGF_TIME_beg, calculated SGF_TIME_end)
   as SESSN_DUR_wQA ,
  INTCK('MINUTE', calculated SGF_TIME_beg, calculated SGF_TIME_end) – 10
   as SESSN_DUR_woQA ,
   ⋮
from work.SASCatalog;
quit;
```

Notice that the calculations of session durations SESSN_DUR_wQA and SESSN_DUR_woQA are identical, with the exception that 10 minutes are subtracted off after the INTCK function for the _woQA version. These two new variables are the number of minutes elapsed during the session (either with or without Q&A). No SAS FORMAT is applied, but you can apply one if it makes sense to do so.

With these conversions and calculations, it's easy to answer some questions about the SASGF2020 Catalog. For example, "How are the Breakout Session speakers split between SAS Employees versus non-SAS employees?"

To answer this question, the SAS program 3_ConvertSAScatalog creates an additional SAS variable, Company_grp, that groups sessions according to whether the primary speaker is a SAS employee or not. Using the two variables Company_grp and SESSN_DUR_wQA a quick run of the FREQUENCY procedure (PROC FREQ), specifically for Breakout Sessions, provides the content in Output 1. This quick analysis indicates that the shorter breakout sessions (30-minute) are evenly split between SAS Employees (SAS) and Users (NotSAS). It also indicates that the longer breakout sessions (60-minutes) are more heavily weighted towards presenters coming from the User (NotSAS) community.



SGF2020 Breakout Sessions, 30- versus 60-minute:
Speaker Split between SAS Employees versus notSAS Employees

Frequency

Table of SESSN_DUR_wQA by Company_grp

	Company_grp		
SESSN_DUR_wQA	NotSAS	SAS	Total
30	90	90	180
60	86	39	125
Total	176	129	305

Output 1 Output from PROC FREQ for SESSN_DUR_wQA * Company_grp

AUTOMATION: CREATING RANGES FOR REPEATABLE QUERYING

The APPENDIX SAS program for this section is 4_RDBMSwAutomation.sas.

The development of concepts against the SASGF2020 Catalog data provides the stepping-stones to solving the following real-world problem.

Background: The data source is an Oracle database. The front end is a procurement system that tracks all purchases of goods from initiation of a requisition into the system through to delivery of all items in a requisition. Various 'events' are tracked as the requisition moves through the procurement system. These events include approvals, generation of purchase orders, status changes, etc.

Problem: You need to analyze these data for bottlenecks in the system. To do this requires data preparation that includes calculations of 'time between events' for several events. Sadly, and for reasons unknown, all dates, times, and datetimes are stored natively as character in this Oracle database. Thus, the calculation of various durations is not possible based on the exiting fields in the database, i.e. you can't perform the math needed for durations on character strings!

Additionally, the data preparation and analysis must be run monthly, each time obtaining all purchase requisitions initiated into the procurement system during the last 12 full calendar months. So, for example, when the process is run in February 2020, only the requisitions that were initiated into the system between Feb 1, 2019 and Jan 31, 2020 should be returned from PROC SQL query to the database.

Solution: The data preparation problem utilizes all the previously presented concepts of INPUT and INTCK, and they are included in the program 4_RDBMSwAutomation. More importantly this program focuses on automating the selection of records in the PROC SQL. For the automation, the AUTOMATIC macro variable &SYSDATE9 is introduced along with SAS function INTNX.

The SAS program assumes a LIBNAME statement connection to the Oracle database is in place, utilizing a *libref* assignment of OraLib1. The Oracle table that houses all the dates needed is called TIDRQMST.

Table 4 shows the Oracle fields names, their renaming in SAS, and the resulting SAS FORMAT after a simple PROC SQL that pulls the fields straight across without conversions. A quick description of each field is provided for context. Notice that all fields are rendered in SAS as FORMAT $W. (character), because that's how the fields are stored in Oracle! The data preparation PROC SQL utilizes the proper use of INPUT to create numeric versions as *EVENTNAME*_dte and the proper use of INTCK to calculate a few of the time between events as *EVENTNAME1*to*EVENTNAME2*_days.

Field in Native RDBMS	Variable in SAS	SAS Format	Field description
REQUISITION_NBR	RNBR	$8.	a unique identifier for the requistion of goods
REQUISITION_LINE	LNBR	$4.	a line counter for each line item in the requisition of goods; COMBINING RNBR and LNBR is the record ID
REQUEST_STATUS	STATUS	$8.	a line level status, tracked to a point in time via STATUS_DATE; use as a record filter
STATUS_DATE	STATUS_DATE	$8.	a date stamp for when a line item status was assigned
DATE_REQUESTED	REQUEST_DATE	$8.	a date stamp for when the requistion was initiated into the procurement system
NEED_DATE	NEED_DATE	$8.	a date stamp for when the line item is needed, set by the REQUESTER
SUGG_PO_DATE	SUGG_PO_DATE	$8.	a date stamp suggested for creating the Purchase Order, based on the NEED_DATE and VENDOR
REQUEST_APPROV_DT	APPROVAL_DATE	$8.	a date stamp for when the line item was approved

ALL dates have been stored natively (on the RDBMS side) as character! However, they are consistently stored in the format YYYYMMDD (which is SAS INFORMAT YYMMDD8.). Need to convert these RDBMS character stored dates to SAS numeric dates in order to apply math related to 'time between events'.

Table 4 Oracle Table OraLib1.TIDRQMST Field Issues

SAS AUTOMATIC MACRO VARIABLE &SYSDATE9

When you start a SAS session the AUTOMATIC macro variable &SYSDATE9 is assigned the date that the session was initiated. It is available throughout the session.

You can view the value assigned to &SYSDATE9 by running a %PUT statement and checking the log:

```
%put sysdate9=&SYSDATE9;
```

Alternatively, you can print it from the view SASHELP.VMACRO using the PRINT procedure:

```
PROC PRINT data=SASHELP.vmacro(where=(name = 'SYSDATE9')) noobs;
run;
```

To automate the date range selection from the Oracle table, this macro variable is used along with SAS function INTNX to obtain the beginning and ending points of the query window used in the where clause of PROC SQL. The where clause is applied to REQUEST_dte, the true numeric temporal component for the DATE_REQUESTED Oracle field. The goal is to grab the last 12 full calendar months prior to &SYSDATE9.

SAS FUNCTION INTNX

The SAS function INTNX is the powerful twin-sister (fraternal of course, not identical) to the previously discussed INTCK function used for the time between events calculations. Just like INTCK, the function INTNX is so powerful and flexible that the reader is encouraged to peruse the online SAS documentation to fully understand it. Only its simplest form is used here. Rather than calculate a time between events, the function INTNX uses a starting point and an interval, along with how many intervals, to move to another place on the date/time continuum and return the value it finds. In this application of the function, INTNX is used to move twelve months prior and one month prior to &SYSDATE9 to form the REQUEST_dte window of the PROC SQL where clause.

In its simplest form, the syntax for the INTCK function is
INTNX(*interval*, *start-from*, *increment* <, '*alignment*'>).

The *interval* parameter is the unit used to move away from *start-from*, such as 'DAY' or 'MONTH' (but there are many more options for interval). The *increment* parameter is directional (can be positive or negative) and indicates how many intervals to move. The *start-from* parameter must be SAS temporal component (literal or variable). The optional *alignment* parameter is used to get the first day of the month for the earliest month, and the last day of the month for the most recent month, in the 12 full calendar month window.

The most critical aspects of the SAS program 4_RDBMSwAutomation are the automation through creation/usage of the two macro variables &begdte9d and &enddte9d. It is assumed that you are familiar with DATA _NULL_, CALL SYMPUTX, PUT, and CATS functions. If you aren't familiar with them, you are missing out! Learn them, they are very useful.

Drilling into the SAS program specifically for the creation of the two macro variables and their usage in PROC SQL looks like this:

```
%global begdte9d enddte9d;
data _null_;
  DTstring="&sysdate9.";
  DTvalue=INPUT(DTstring,anydtdte.);
  call SYMPUTX('begdte9d', CATS("'",
               PUT(INTNX('month', DTvalue, -12, 'begin'), DATE9.), "'d"));
  call SYMPUTX('enddte9d', CATS("'",
               PUT(INTNX('month', DTvalue, -1, 'end'), DATE9.), "'d"));
run;
%put &begdte9d &enddte9d; /* check values in the log */

PROC SQL;
  create table work.TIDRQMST as
  select
  ⋮
  from OraLib1.TIDRQMST
  where  &begdte9d <= calculated REQUEST_dte <= &enddte9d
  ;
quit;
```

CONCLUSION

This paper begins with motivation for learning the ins and outs of working with the SAS temporal components of type date, time, and datetime. To keep things simple and repeatable, most of the concepts are introduced in relationship to the SAS Global Forum 2020 schedule, available at the conference website as an Excel file download. SAS literals, INFORMATS and FORMATs for temporal components are discussed and demonstrated, before moving into some of the challenges of working with SAS variables as they are fed from sources like Excel and RDBMS. The SAS functions INPUT, INTCK and DHMS are the keys to the initial data preparation magic. Lastly there is a sneak peek on where you can take this against RDBMS, including automation via &SYSDATE9 and SAS function INTNX. With a little effort you will master these concepts until: Time is on your side. Yes, it is.

ACKNOWLEDGMENTS

I extend my gratitude to my current leadership at the Idaho National Laboratory for supporting my pursuit of career development through conference participation at SAS Global Forum. I am indebted to my current SAS Representative, Pamela Tomski, for being the inspiration that keeps me engaged in the SAS Community, including forming and leading the INL SAS Users Group and continuing to share my SAS knowledge with others.

Additionally, I thank the 2020 Conference Chair, Lisa Mendez, and her team for accepting this abstract and paper to the SAS Global Forum 2020 and providing me the opportunity to present the work.

RECOMMENDED READING

- *SAS® 9.4 Functions and CALL Routines: Reference*
- *SAS® 9.4 Formats and Informats: Reference*
- *Carpenter's Guide to Innovative SAS® Techniques*
- *Carole A. Jesse, "Romancing Your Data: The Getting-to-Know-You-Phase", Proceedings of the SAS Global Forum 2011 Conference (Cary, NC: SAS Institute Inc., 2011).*

CONTACT INFORMATION

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

Author:	Carole Jesse
Email:	Carole.Jesse@inl.gov
Twitter:	http://www.twitter.com/CaroleJesse
LinkedIn:	https://www.linkedin.com/in/carolejesse
GitHub:	https://github.com/CaroleJesse
	the SAS programs for this paper are in GitHub repository SASGF2020
	the Excel file (the actual version used) is also in the GitHub repository

APPENDIX

```
/******************************************************************************
1_SASliterals.sas
SAS Global Forum 2020, paper #4801, by Carole Jesse
Uses BASE SAS

Using the date/time of SASGF2020 paper #4801 to understand:
SAS literals for date, time and datetime
SAS formats for date, time and datetime
******************************************************************************/

/* Create a dataset related to date/time of SASGF2020 paper #4801 */
data work.SASGF2020_4801;
 Date='01APR2020'd;
 Time='10:00:00't; /* 24-hour clock! 10 PM would be 22:00:00 */
 DateTime='01APR2020:10:00:00'dt;
run;
PROC CONTENTS data=work.SASGF2020_4801 order=varnum;
run;

/* Create a dataset related to date/time of SASGF2020 paper #4801 */
/* Apply SAS F-ormats at data set creation */
data work.SASGF2020_4801F;
  Date='01APR2020'd;
  Time='10:00:00't;
  DateTime='01APR2020:10:00:00'dt;
  format  Date YYMMDDp10.  Time TIMEAMPM8.  DateTime DATETIME19.;
run;
PROC CONTENTS data=work.SASGF2020_4801F order=varnum;
run;
/*********************/
/**** End Program ****/
/*********************/


/******************************************************************************
2_ReadSAScatalog.sas
SAS Global Forum 2020, paper #4801, by Carole Jesse
Uses BASE SAS, SAS/ACCESS to PC Files

Obtain the SASGF2020 catalog at http://sasgfsessioncatalog.com/#/search/
and clicking the 'Download all sessions in an Excel spreadsheet'.
The result is a single tab Excel file called SAScatalog.xlsx.
The data tab is called 'Full Catalog'.
The data range (including header) is A2:AB695 (as of Feb 15, 2020, there are
693 data observations in the catalog).

&DataDir is a macro variable containing the path to the downloaded file:
%let DataDir = YourFilePath\;
******************************************************************************/
%let filenme1=SASCatalog.xlsx;

* PROC IMPORT (SAS/ACCESS to PC Files) using Tabname$Range syntax;
PROC IMPORT OUT= work.SASCat
            DATAFILE= "&DataDir.&filenme1"
            DBMS=XLSX REPLACE
;
```

```
RANGE="Full Catalog$A2:AB695";
GETNAMES=YES;
RUN;

* PROC SQL (although data step will work too), select cols, cleanup names;
PROC SQL;
create table work.SASCatalog as
select
'Session Title'n as Title ,
Description as Abstract ,
'Session Type'n as Type ,
Topic ,
Industry ,
'Primary Product'n as PrimaryProduct ,
'Job Role'n as JobRole ,
'Skill Level'n as SkillLevel ,
Day ,
'Start Time'n as StartTime ,
'End Time'n as EndTime ,
Room ,
'Primary Speaker First Name'n as SPKR_Fname ,
'Primary Speaker Last Name'n as SPKR_Lname ,
'Primary Speaker Company'n as SPKR_Company ,
'Primary Speaker Job Role'n as SPKR_JobRole
from work.SAScat
;
quit;

PROC CONTENTS data=work.SASCatalog order=varnum ;
run;
* temporal components Day, StartTime, EndTime are character;
/*********************/
/**** End Program ****/
/*********************/


/*****************************************************************************
3_ConvertSAScatalog.sas
SAS Global Forum 2020, paper #4801, by Carole Jesse
Uses BASE SAS

Convert character forms of the variables to numeric SAS DATE, TIME, DATETIME
Create calculated variables

NOTE:
Input data work.SASCatalog is output from 2_ReadSAScatalog.sas
*****************************************************************************/
/* Use PROC SQL (although data step will work too):
transform character variables to true numeric: utilize INPUT()
create new Variables and Calculations: utilize DHMS(), INTCK() */
PROC SQL;
create table work.SASCatalog2 as
select
Title ,
Type ,
Topic ,
Industry ,
PrimaryProduct ,
JobRole ,
SkillLevel ,

Day ,
/* Convert char Day to num SGF_DATE, use SAS format MMDDYYs10. */
INPUT(Day,ANYDTDTE10.) as SGF_DATE  format=MMDDYYs10. ,

StartTime ,
/* Convert char StartTime to num SGF_TIME_beg, use SAS format TIMEAMPM8. */
INPUT(StartTime,ANYDTTME8.) as SGF_TIME_beg  format=TIMEAMPM8. ,

EndTime ,
/* Convert char EndTime to num SGF_TIME_beg, use SAS format TIMEAMPM8. */
INPUT(EndTime,ANYDTTME8.) as SGF_TIME_end  format=TIMEAMPM8. ,

/* Use the SAS function DHMS() to combine SGF_DATE and SGF_TIME_beg into
SGF_DATETIME
syntax to combine a legit SAS date with a legit SAS time is DHMS(SAS date, 0,
0, SAS time) */
DHMS(input(Day,ANYDTDTE10.),0,0,input(StartTime,ANYDTTME8.)) as SGF_DATETIME
format=DATETIME21. ,

/* ALTERNATE for SGF_DATETIME creation */
DHMS(calculated SGF_DATE,0,0,calculated SGF_TIME_beg) as SGF_DATETIME_alt
format=DATETIME21. ,

/* Use SAS function INTCK() to calculate Session Durations, w & wo the 10
minute Q&A window:
syntax is  INTCK(interval <multiple> <.shift-index>, start-date, end-date,
<'method'>) */
INTCK('MINUTE',calculated SGF_TIME_beg, calculated SGF_TIME_end)      as
SESSN_DUR_wQA ,
INTCK('MINUTE',calculated SGF_TIME_beg, calculated SGF_TIME_end) - 10 as
SESSN_DUR_woQA ,

/* Primary Speaker info */
SPKR_Fname ,
SPKR_Lname ,
SPKR_Company ,
SPKR_JobRole ,

/* grouper for the Speaker Company */
CASE WHEN substr(SPKR_Company,1,3) = 'SAS' then 'SAS'
      ELSE 'NotSAS' end as Company_grp format=$varying6.

from work.SAScatalog
;
quit;
/*********************/
/**** End Program ****/
/*********************/


/****************************************************************************
4_RDBMSwAutomation.sas
SAS Global Forum 2020, paper #4801, by Carole Jesse
Uses Base SAS and SAS/ACCESS to ODBC for Oracle connection
```

```
Combine all the concepts against RDBMS
Add automation via automatic macro variable &SYSDATE9 and INTNX function
To:
Go against a RDBMS database where dates are stored as character, convert them
When the program runs it should automate getting
the last 12 full calendar months of data, based on the SAS session date
******************************************************************************/
/* create macro vars &begdte9d and &enddte9d, for PROC SQL where clause */
%global begdte9d enddte9d;
data _null_;
DTstring="&sysdate9.";
DTvalue=INPUT(DTstring,anydtdte.);
/* NOTE: DTvalue is the SAS date numeric */
/* create macro var begdte9d, the first day of the first month in the window
of 12 months*/
call SYMPUTX('begdte9d', CATS("'", PUT(INTNX('month', DTvalue, -12, 'begin'),
DATE9.),"'d"));
/* create macro var enddte9d, the last day of the last month in the window of
12 months*/
call SYMPUTX('enddte9d', CATS("'", PUT(INTNX('month', DTvalue, -1, 'end'),
DATE9.),"'d"));
run;
%put &begdte9d &enddte9d;

/* PROC SQL using Oracle connection with libref OraLib1 */
/* correct char dates with read of Oracle table */
/* calculate time between events */
PROC SQL;
create table work.TIDRQMST as
select
REQUISITION_NBR as RNBR ,
REQUISITION_LINE as RLINE ,
/* char date conversions with INPUT */
INPUT(STATUS_DATE,YYMMDD8.) as STATUS_dte  format=MMDDYYs10. ,
INPUT(DATE_REQUESTED,YYMMDD8.) as REQUEST_dte  format=MMDDYYs10. ,
INPUT(NEED_DATE,YYMMDD8.) as NEED_dte  format=MMDDYYs10. ,
INPUT(SUGG_PO_DATE,YYMMDD8.) as SUGG_PO_dte  format=MMDDYYs10. ,
INPUT(REQUEST_APPROV_DT,YYMMDD8.) as APPROV_dte  format=MMDDYYs10. ,
/* time between events with INTCK */
INTCK('DAYS',calculated REQUEST_DTE, calculated NEED_DTE) as REQtoNEED_days ,
INTCK('DAYS',calculated REQUEST_DTE, calculated APPROV_DTE) as REQtoAPPR_days
,
INTCK('DAYS',calculated REQUEST_DTE, calculated SUGG_PO_DTE) as
REQtoSUGGPO_days

from OraLib1.tidrqmst
/* 12 full cal months, prior to SAS session &SYSDATE9 */
where &begdte9d <= calculated REQUEST_dte <= &enddte9d
;
quit;
/*********************/
/**** End Program ****/
/*********************/
```