

Paper 228-27

Making SAS® Dates From All Your Character Dates...**No Pixie Dust Required**

Jeanina Worden, PRA International, Lenexa, KS

ABSTRACT

Within a clinical database, the need to capture exactly what is on a document (partial dates, "12DEC2000", or "Unknown" as a comment) requires the field attribute for all dates to be set as character. However, the client will typically request the final datasets to include an additional SAS® formatted, numeric date field for these dates. Though not a difficult task, using INPUT and a SAS format, it can be a VERY time consuming one. Given that when doing billable work the old saying is true, "time is money", there is a need to decrease programming time by decreasing repetitive tasks. Here lies a perfect opportunity to put a macro to work.

The macro discussed here, developed with V6.12, creates corresponding numeric SAS date fields for all the character date fields within a database. Using the SQL dictionary table COLUMNS, the date variables are identified by specifying LIKE '%DATE%'. Then CALL SYMPUT statements create variables for use within the macro. The macro contains two LOOP statements and a DATA step that add a formatted date variable by using an IF statement and INPUT. The updated datasets are placed in a new library to ensure the structures of the raw datasets are maintained.

BEFORE WE GET STARTED

Before getting started describing the macro revealed herein, it might help to understand what I do that causes creativity and the power of SAS to be used to their fullest. My position at PRA International Inc., a pharmaceutical research company, entails creation of databases in a Clinical Data Management system that creates Oracle tables. I then convert those tables to SAS datasets using base SAS v.6.12. These datasets are then modified to specific structure criteria for the final output. The criterion may be different from project to project, causing each dataset to be modified individually, rather than plugged into a standard model. However, there is one standard modification, changing character dates into SAS dates, which must be done multiple times within each dataset of each database. Given that this must be done in billable time this repetition can be costly for the client. Thus, to be more efficient with programming is the reason for this macro and this paper.

EVERYBODY WANTS TO CHANGE THE WORLD

By definition one of the reasons to create a macro is to eliminate repetition in your programming, so it was decided to create a macro to convert all character dates into SAS dates. However, out of all the fields of raw data, how does one know which fields are date fields? If this macro is for use by only one person, or the sole user of the macro created the raw data fields, then there is no problem. The same field names can be used each time a date field is created in the original database. For example, if dates of birth will always be DOB and dates of contact will always be VISITDT, then the macro will loop through looking for these "standard" fields. However, this is not very realistic or practical. Unfortunately, at some point one will come to realize the world will not conform to specific naming conventions in order to accommodate one macro. So, how does one get the field names for every date field? As with this macro, one could start by using the SQL DICTIONARY table COLUMNS. The SQL dictionary tables are tables that can be used to retrieve information about

elements associated with the current SAS session. Before the macro starts off, a table is created of only the date fields by using the dictionary table COLUMNS to get the module name, field name, type, length and label where the label contains the word "DATE".

```
proc sql;
create table dateflds as
select libname, memname, name, type, length,
       label
from dictionary.columns
where libname='TEMP' and memname ne 'PATINFO'
and upcase(label) like '%DATE%';
quit;
```

It only pulls this information from the library "TEMP" since that is where the raw datasets were stored. The TEMP datasets will not be modified in any way to ensure their integrity; instead a new library TEMP2 will be used to store the modified datasets. The WHERE statement also uses LIKE to allow for label variations, this will capture labels such as "Date of Birth" and "Event Date". A caveat that should be mentioned is that this WHERE statement will also sort out variables with labels such as "Was a Visit Date required?" The macro will try to create a date from the YES/NO response. However, these fields will remain blank and can be deleted later. Since the string is case sensitive it is a good idea to use UPCASE on the label, again to allow for variations.

FIRST THINGS FIRST – MACRO VARIABLES

The first thing is actually to tell the system your starting a macro simple by using %macro followed by a distinct name that meets the standard SAS naming conventions. For this macro the name is cdt_ndt, so it starts like this:

```
%macro cdt_ndt;
```

Any code between this and the %mend defines the macro. Now, the DATEFLDS table that was created can be used to create macro variables that will be used to name the converted variables and determine the number of times each loop is processed within the macro. There are two types of macro variables used within this macro, user defined and system created. First, for the user defined, the scope of the variables must be defined as LOCAL or GLOBAL. If it is GLOBAL, a macro variable exists for the remainder of the current SAS session, making it available to any part of the SAS session. Since we only need the variables for use during execution of the macro, both variables are LOCAL, available only until the macro stops executing.

```
%local i;
```

```
%local j;
```

Next, as explained before, the module and variable names may not be constant across multiple projects. Therefore, CALL SYMPUT statements are used to create macro variables from the new DATFLDS table within DATA steps.

```
data _null_;
set dateflds end=fnl;
call symput('module' || left(_n_),
           trim(memname));
if fnl then call symput('max',_n_);
run;
```

```
data _null_;
set dateflds end=last;
call symput('var' || left(_n_),trim(name));
call symput('datf' || left(_n_),
           trim(name) || 'n');
```

```
if last then call symput('count',_n_);
run;
```

By using `_NULL_`, the SAS System executes the DATA step without creating a SAS data set. This can be a more efficient use of computer resources since the dataset does not need to be stored. The new macro variables that are produced: `max`, `count`, `module`, `var`, and `datf` are followed by `_n_`. `_n_` is an automatic variable that is created by the SYMPUT statement and added to the program data vector but not output to the dataset. It initially is set to 1 then will increment by one each time the statement is initiated, or in this case, each time a variable is created. This will allow all variables to be stored, otherwise it will only store the last one created.

HERE WE GO LOOP-DEE-LOOP

Now that the variables have been created and the values stored in the system, two loops are used to do the actual converting. To understand how it works, picture the wheels inside a clock; there is a big one that moves the hour hand, which moves once only after the smaller one has moved around the minute hand sixty times. For each time the first loop cycles to the next dataset, the second loop moves through each variable within that dataset verifying whether it's a date field and converting the ones that are. For the first loop, "i" is assigned as 1 to MAX, which was defined in the above DATA step as the total number of memnames in DATEFLDS.

```
%do i=1 %to &max;
data temp2.&&module&i;
set temp.&&module&i;
```

The new datasets will be stored into TEMP2 to maintain TEMP in its original state, as mentioned previously. Within each loop through the datasets, the second loop uses "j", which is assigned as 1 to COUNT. COUNT was defined in the second DATA step above as the total number of names in DATEFLDS.

```
%do j=1 %to &count;
format &&datf&j date9.;
%if variable=&&var&j %then;
&&dat&j=input (&&var&j, date9.);
```

Within this second loop an IF-THEN statement is used to determine if the current variable is one of the date fields assigned to VAR, then INPUT and the DATE9 format are used to create the corresponding, SAS formatted, date variable stored in DATF. The name of the DATF variable is the same as the original variable followed by an "n" to denote it's numeric characteristic. The library TEMP2 will have all the datasets and variables as the original TEMP dataset, as well as additional fields of corresponding SAS dates for each date field. All programs to create the final datasets can then use these modified datasets without having to convert dates again.

AND...ACTION

The macro is now complete, or at least ended with `%mend`. However, for it to run it must be called into action. In this program the call is included following the closing of the macro.

```
%mend cdt_ndt;

options mprint mlogic;

%cdt_ndt;
```

Simply by `%macro-name`, in this case `%cdt_ndt`, the macro is initiated. While sometimes this call statement be included in an autoexec program so that it was run with each update. This macro and call statement are self contained program to maintain control over the newly created library, TEMP2.

THE BIG SUM UP

This paper has been all about using a macro to get corresponding SAS dates for all the date fields in a raw database. While the conversion itself isn't difficult using INPUT and a SAS

format, like DATE9., it can be time consuming if you have to do this conversion multiple times within a program and throughout the modifications to datasets. First the SQL dictionary table COLUMNS was used to sort out the date fields within the raw database. Then macro variables are created as LOCAL, user defined variables as well as system assigned variables. The `_NULL_` dataset is used for each DATA step since storage of these datasets is not necessary. These variables are then used within two loops; one nestled inside of the other. The first loops through the original datasets and setting them into a new library in order to maintain the integrity of the original data. The second looping through the variables within each dataset looking for the date fields identified from the COLUMNS table. When a date field is encountered it then uses INPUT and DATE9 to create a corresponding, SAS formatted date field with the same name as the original field with the addition of "n" to denote it's numeric property. The final step is to call the macro, within the program to initiate it's running.

REFERENCES

SAS Institute Inc. (1990), "SAS Language Reference, Reference Version 6, First Edition," Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1990), "SAS Procedures Guide, Version 6, Third Edition," Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1997), "SAS Macro Language Reference, Reference Version 6, First Edition," Cary, NC: SAS Institute Inc.

SAS Institute Inc. (1990), "SAS Guide to Macro Processing, Version 6, Second Edition," Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

A big thanks to Theresa Bosserman for her excellent proofreading skills and putting up with my numerous requests for rereads.

CONTACT INFORMATION

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

Jeanina Worden
PRA International Inc.
Phone: 913-577-2883
Email: wordenjeanina@praintl.com