Paper 043-31

Question:
How Do I Find Out What That _TYPE_ Value Is from My MEANS Procedure?
Answer: The FINDTYPE Macro!

Dan Bruns, Chattanooga, TN

ABSTRACT

Have you ever been really confused by the _TYPE_ variable with PROC MEANS or PROC SUMMARY? Have you ever had trouble determining the values of _TYPE_? The FINDTYPE macro is your answer! Accepts variable names as parameters and returns the corresponding _TYPE_ value. A tip worth ten minutes of your valuable conference time!

CREDIT TO THE MACROS AUTHOR

I always thought this would be a very powerful tool to code, but never did. A good friend of mine that I have done some work for over the past few years is the author of this macro — Ron Coleman. Although he now works for SAS, the many training materials he developed over the years lives on and the Macros course includes this powerful tool. I want to thank Ron for allowing me to present to you this very powerful macro tool.

FRUSTRATION

Since the dawn of SAS the MEANS and SUMMARY procedures have posed a frustrating challenge to SAS programmers. After you generate an output SAS dataset you have the challenge in subsequent steps to subset it to just the summary observations you want.

```
Proc means data=_______________;
   class  var1 var2 var3 var4 ;
   var x y ;
   Output  out=work.sum1 ;
   Run;
Data work.newsum;
   Set work.sum1 ;
   If _type_ = 5 ;
   run;
```

Most programmers I know simply do a PROC PRINT after the PROC MEANS, find the _TYPE_ value, then go back and plug it in the code. NO MORE!

```
Data work.newsum;
   Set work.sum1 ;
   If _type_ = %FINDTYPE(work.sum1, var2 var4);
   run;
```

THE MACRO CODE

```
%macro findtype(table,vars);
   %local index fid flag target;
   %let fid = %sysfunc(open(&table,i));
   %if not &fid %then %goto exit;
   %let vars = %upcase(&vars);
   %let flag = 0;
   %let _type_ = 0;
   %do i = %sysfunc(attrn(&fid,nvars)) %to 1 %by -1;
      %let target = %upcase(%sysfunc(varname(&fid,&i)));
```

```
      %if &flag %then %do;
         %let index = 1;
         %do %while (%length(%scan(&vars,&index)) > 0);
            %if &target = %scan(&vars,&index) %then
               %let _type_ = %eval(&_type_ + 2**(&flag - &i - 1));
            %let index = %eval(&index + 1);
         %end;
      %end;
      %else %if &target = _TYPE_ %then
         %let flag = &i;
   %end;
   %let fid = %sysfunc(close(&fid));
   &_type_
   %goto term;
%Exit:;
   -1
%term:
%mend findtype;
```

LET'S LOOK AT EACH LINE ONE AT A TIME.

`%macro findtype(table,vars);`
Starts the **findtype** macro definition. Two parameters are required: **table,** which dataset the variables are in, and **vars**, which variables summary you are looking for.

In our example above table is **work.sum1** and vars is **var2 var4**.

`%local index fid flag target;`
Defines which macro variables are only known within this macro (local).

`%let fid = %sysfunc(open(&table,i));`
Use the macro function %**sysfunc** to invoke the dataset function **open** to open the dataset name (passed to the macro in the **table** variable) for input and define a unique file identifier (**fid** macro variable**)** for it. This **fid** is required for use in other dataset functions to be performed on this dataset later in the macro.

`%if not &fid %then %goto exit;`
If the dataset is not found (fid = 0), go to an error exit.

`%let vars = %upcase(&vars);`
Uppercases the variable names passed to the macro in the **vars** macro variable. (For ease of comparison later.)

`%let flag = 0;`
Set the macro variable **flag** to 0. **Flag** is used to identify which variable the actual _TYPE_ variable is in the dataset.

`%let _type_ = 0;`
Set the macro variable _**type**_ to 0. _**type**_ is used to return the actual _type_ value back to where the macro was called/invoked.

`%do i = %sysfunc(attrn(&fid,nvars)) %to 1 %by -1;`
Perform the statements following this **%do** (down to its corresponding **%end** statement). The %**sysfunc** macro function is used to invoke the dataset function **attrn** to find the number of numeric variables in the **fid** dataset. So, this value is then used as the **starting point** for the iterative do-group that will then increment the **i** macro variable by -1 until it reaches a value of 1. The incrementing is being done in reverse order for a very specific and important reason, the variables are arranged in the output dataset with the BY variables listed first, the CLASS variables, the _TYPE_, _FREQ_, and _STAT_, any ID variables, and finally the output statistics. And this macro looks for the _TYPE_ variable to know where the CLASS variables begin **AND compute the _type_ value**.

In our example above the value for the macro variable **i** will start at **9**.

`%let target = %upcase(%sysfunc(varname(&fid,&i)));`
Use the macro function %**sysfunc** to invoke the dataset function **varname** to get the name of the **i-**th variable from the **fid** dataset. Upcase it and store it in the **target** macro variable.

`%if &flag %then %do;`
If the value of the **flag** macro variable is not zero (the _TYPE variable is found), perform the following code to the corresponding **%end** statement.

`%let index = 1;`
Set the macro variable **index** to 1. **Index** is used to identify the dataset variable name from the list in the **vars** macro variable (1 – first name, 2 – second name, etc).

`%do %while (%length(%scan(&vars,&index)) > 0);`
Perform the statements following this **%do** (down to its corresponding **%end** statement) while the %**scan** macro function returns a value – a dataset variable name from the list in the **vars** macro variable, depending on the value of the **index** macro variable (1 – first name, 2 – second name, etc).

`%if &target = %scan(&vars,&index) %then`
If the **target** macro variable equals the value returned from the %**scan** macro function – a dataset variable name from the list in the **vars** macro variable, depending on the value of the **index** macro variable (1 – first name, 2 – second name, etc), perform the next statement.

`%let _type_ = %eval(&_type_ + 2**(&flag - &i - 1));`
Here it is folks…..the statement that actually computes the **_type_** value.

In our example above we specified 4 variables on the CLASS statement and would expect to find the variables in the dataset ordered as: var1 var2 var3 var4 _TYPE_ _FREQ_ _STAT_ x y.

Using a call of: %FINDTYPE(work.sum1, var2 var4) we would expect to calculate based on the positions of the variables var2 (second) and var4 (fourth).

> **&flag has a value of 5 (the order sequence of _TYPE_)**
> **When &i = 4 the calculation of &_type_ resolves to: %eval(0 + 2**(5-4-1)) or 0+2**0 or 1**
> **When &i = 2 the calculation of &_type_ resolves as: %eval(1 + 2**(5-2-1)) or 1+2**2 or 5**
> **Giving &_type_ the value of 5 to return.**

`%let index = %eval(&index + 1);`
Increments the **index** macro variables value by 1.

`%end;`
Ends the **%do %while** do-group.

`%end;`
Ends the **%if &flag %then %do;** do-group.

`%else %if &target = _TYPE_ %then`
`%let flag = &i;`
This is the **%else** to the **%if &flag** do-group. So, if the value of the **flag** macro variable IS zero, check if the **target** macro variable is the actual _TYPE_ variable from the summary dataset; if so, set **flag** to the value of the **i** macro variable.

In our example above the **flag** macro variable will eventually have a value of **5**.

`%end;`
Ends the **%do i =** do-group.

%let fid = %sysfunc(close(&fid));
Use the macro function %**sysfunc** to invoke the dataset function **close** to close the dataset identified by the unique **fid** opened earlier.

&_type_
Returns the value of the **_type_** macro variable from whence this macro was called/invoked.

%goto term;
Branch to the end of the macro.

%Exit:;
-1
Returns a -**1** from the macro invocation…..an error indicator. (Also an invalid _type_ value.)

%term:
%mend findtype;
Macro end/exit.

DISCLAIMERS

1. This version of the FINDTYPE macro does not properly handle **By variables** that are used in the creation of the dataset. Use an appropriate WHERE clause for BY values.

2. The variable ordering referred to equates to the NVAR value found in PROC CONTENTS not the NPOS! Beginning in SAS V8.2 all numeric variables are physically positioned at the beginning of an observation for access efficiency. NPOS (physical position with the observation) and NVAR (logical ordering of variables) cannot be assumed to be the same.

IN SUMMARY

The macro was designed to be used in IF and WHERE statements but I am sure some ingenious SAS programmer somewhere will find another use for it someday.

This powerful macro from Ron Coleman has been in my SAS toolbox for years. I have shared it with the SAS users in my company and many others and still get email thanking me for it. I hope you find it as useful as I have and share it with all your SAS friends.

So good luck!!!

ACKNOWLEDGEMENTS

PAPER AUTHOR

If you have any questions or comments, please write or call:

Dan Bruns
Tennessee Valley Authority
1101 Market Street (MP 2B)
Chattanooga, TN 37402
423/751-6430 Fax 423/751-3163
Email: debruns@tva.gov