Paper 034-2007

Hurry!!!, Hurry!!! Step Right UP. Use The "Magical Compound Where Clause" to Eliminate Data Steps, Reduce Processing Steps, Speed Job Turnaround, and Mystify Your Friends.

William E Benjamin Jr, Phoenix, AZ

ABSTRACT

The "Magical Compound Where Clause" can be used by beginning or advanced level programmers on the input side of a Base SAS® or SAS procedure dataset reference on any computer operating system. This is especially useful if extracting subsets of records from large files. When working with "large disk files" that may take over an hour to read, programmers need creative ways to cut job steps and save time. When many programmers face the task pulling some records from a file for a report or new analysis, most will run a simple SAS Data step then use PROC Sort to reorder the file for the next step. AH!!!, But the "Magical Compound Where Clause" can eliminate the SAS Data Step altogether. Saving time because only the "Magical Compound Where Clause" variables are read before selecting the records to be sorted. This paper will introduce four advanced command structures that the "Magical Compound Where Clause" can use on the input side of dataset options. These options save time and steps when processing files large or small. Many programmers are familiar with the often used "KEEP=" and "DROP=" commands when inputting data to a SAS "SET" command. But, can you convert variable types, check a list and test for a variable value all in the same "Magical Compound Where Clause", as you send data to a PROC Sort operation (without a data step?)? You Bet You Can!!! This paper will show you how to optimize your programs and mystify your friends.

INTRODUCTION

The "Magical Compound Where Clause" is probably one of the most misunderstood and under used features of the SAS dataset options. Of course SAS does not call it "Magical", it is only "Compound" when the user goes out of their way to combine options, and it is really the "**WHERE**=" clause of the dataset options, and the technique could be applied to the Data Step "**WHERE**" clause too. Now that the hype about the "Magical Compound Where Clause" has been exposed, and the reader hooked, the details will come forth.

THE BASICS – THE CODE SYNTAX

Any programmer that has worked with just about any programming language has seen a syntax drawing. The SAS Help system has a syntax definition for just about everything, including **SET** statement or the **SORT** Procedure. Notice the syntax of the **SET** statement listed here:

```
SET<SAS-data-set(s) <(data-set-options(s) )>> <options>;
```

And the simple form of the Sort Procedure:

```
PROC SORT DATA=SAS-data-set;
       By key;
```

Everyone has seen and used these, if you have worked with SAS for more than a day. Well, one of the "**<(data-set-options(s))>**" is the "**WHERE**=" option. Note the syntax of it from the SAS Help system.

=-

Syntax

WHERE=(where-expression-1<logical-operator where-expression-n>)

Syntax Description

where-expression

is an arithmetic or logical expression that consists of a sequence of operators, operands, and SAS functions. An operand is a variable, a SAS function, or a constant. An operator is a symbol that requests a comparison, logical operation, or arithmetic calculation. The expression must be enclosed in parentheses.

logical-operator

can be AND, AND NOT, OR, or OR NOT.

=-

It looks simple doesn't it, but if it does that means you're just not looking at it close enough. That description uses 49 words to describe the "where-expression" in that syntax chart, and you can include "n" of them. How big is "n"? They don't say. Look, you can include "arithmetic" or "logical" expressions or operands, "SAS functions", a "variable", a "constant" or symbols that request .... Well you get the picture, it goes on and on.

BUT WHAT CAN IT DO FOR YOU?

This is where the paper takes over. The SAS "**WHERE=**" clause gains its power from its flexibility. The simple case of **WHERE=(variable_a = value_b)** is often used by most SAS programmers. The following is the first of four slightly more complex examples:

EXAMPLE 1 – (COMPOUND WHERE-EXPRESSION USING SAS FUNCTIONS)

You have a file with 3 million records and 1850 variables, you need a file for the director at 10:00 AM tomorrow that contains all active customers from Ohio that have bought wagons or bicycles in the last year. You need all of the variables because the director is going to use the file to predict who might buy sideboards for the wagon or a tow bar to connect the wagon to the bicycle. The data file is not on the same computer as the one running the SAS job and network traffic will slow your job to the point that you can read the data file once in the time allotted. The file needs to be sorted by ZIPCODE, CUSTOMER, and
ITEM (wagon or bicycle), and the people who entered the data were not consistent in the use of the shift key (Ohio was also spelled OhiO and OHIO). Note the code below.

```
(1) PROC SORT DATA=Store.inventory
(2)        (WHERE=
(3)                  (
(4)                       ("OHIO"=(UPCASE(state)))
(5)                       and
(6)                       (UPCASE(item) in ('WAGON', 'BICYCLE')))
(7)                   )
(8)        )
(9)               OUT=My.new_list
(10)       ;
(11)       BY Zip_code Customer Item;
(12)    RUN;
```

The author chose to write this call to **PROC SORT** on 10 lines to explain it more easily and also to show the tests completely enclosed in parentheses, to make it clear where logical breaks are expected.

Line 1. This is the call to **PROC SORT** with the input DATA file named store.inventory.

Line 2. All dataset options are enclosed in parentheses, the first starts the dataset options. Then the beginning of the "**WHERE=**" is shown.

Line 3. This parenthesis opens the Object of the "**WHERE=**" clause.

Line 4. This is the test for the name of the State. The "**UPCASE**" function in this case does not modify the contents of the variable STATE, but does convert the value to uppercase letters for the test against the value "OHIO", which is in all capital letters in this example.

Line 5. This operator indicates that both Line 4 and Line 6 must be true for the record to be accepted.

Line 6. Once again the "**UPCASE**" function is used to ensure that the test for the value being in the list 'WAGON' and 'BICYCLE' is uniform against all upper case letters.

Line 7. The parenthesis on this line and line 3 group the object of the **WHERE=** clause.

Line 8. The parenthesis on this line and line 2 group the dataset options.

Line 9. Identifies the output file name of the sorted file, this is an optional sort output file that preserves the original file. This is used here because only a subset of the original file is being sorted. Leaving this option off of this sort will reduce the input file to only the selected records. Reducing the size of the input file may not be the intended result of this task.

Line 10. This is the end of the **PROC SORT** command.

Line 11. This is the **BY** clause of the **PROC SORT** procedure.

Line 12. In a stream of code this line could be optional but as the last line of a PC SAS job it is required to get the job finished. The **RUN** command also will write the SAS notes before the start of the next code step.

EXAMPLE 2 – SIMPLE WHERE-EXPRESSION USING SAS LOOK-UP TECHNIQUES

When was the last time that you were given list of 100 or 100,000 records to pull from a big file, only to find out you could not sort the big file in time and your list did not have the variables to be sorted into the order of the large file. Then you had to sort the output file into a new order too. Worst of all your boss sent the list in a *.TXT file, and the big file has numeric keys. Looks like more than one pass, right? ... Wrong! It can all be done in one pass of the big file.

MACRO VARIABLE LIST

If your list is small your **WHERE=** clause would be simple, create a macro variable, and embed it into the **WHERE=** clause, like so...

```
%let   customers =   '11111','22222','33333','44444','55555',
                     '66666','77777','88888','99999','10101';

DATA My.out_file;
set My.in_file       (WHERE=(customer_number in (&customers));
run;
```

But if customer_number is numeric then just make a small change and it would look like this, both of the operands here are character:

```
DATA My.out_file;
set My.in_file       (WHERE=(put(customer_number,z5.) in (&customers)));
run;
```

EXAMPLE 3 – CREATING A FORMAT FILE FOR THE TABLE LOOK-UP

This example is a little more complex than the second example. It is a three-step process. First, convert the input list into a form that can be used by **PROC FORMAT** directly to create a **SAS FORMAT** that identifies all records to be selected. Large lists used to be a problem, but with computers that have multi-gigabytes of RAM memory storage, the size of a format is now not usually a problem. (if you are not using an IBM Mainframe)

Therefore, if your boss only sends you the location of the SAS file with 100,000 records then try this trick (limited only by the amount of memory available at run time):

```
(1)    DATA fmt;
(2)           Set  Boss_list (keep=customer_number) END=eof;
(3)           START = PUT(customer_number,z12.);
(4)           LABEL = 'Y';
(5)           FMTNAME = 'onlist';
(6)           OUTPUT;
(7)           IF (eof = 1) THEN do;
(8)                  START = 'other';
(9)                  LABEL = 'N';
(10)                 OUTPUT;
(11)          End;
(12)
(13)   PROC FORMAT CNTLIN=fmt;
(14)   RUN;
(15)
(16)   PROC SORT DATA=big_file
(17)          (WHERE=((PUT(customer_number,onlist.) = 'Y')))
(18)          OUT=new_file;
(19)          BY customer_number;
(20)
(21)   RUN;
```

This code makes one pass over the small file of customer numbers and creates a **SAS FORMAT** for the **SORT** routine to use on the input side of the processing. As it reads the records in the big_file the customer_number is tested to see if it is on the list of required records and when found it is added to the group of records to be sorted. A detailed explanation follows.

Line 1. This is the definition of a temporary dataset which will be used as input to **PROC FORMAT**. This dataset will contain the customer numbers to be selected from the output file. Each record will also have a label and a format name variable.

Line 2. This **SET** statement would point to the input dataset with the list of customer numbers, a flag variable called EOF is created to indicate when the last record has been read from the input file.

Line 3. This line places the customer number (a numeric variable customer_number from the input file) into the variable called START which is used by **PROC FORMAT** to build the format. The output variable START in this case is a character variable, be aware that **PROC FORMAT** allows a special case of "other" as a default for any value not on the list, so the variable START is defined here as more than 5 characters.

Line 4. This line places the value "Y" into the variable called LABEL, which is used by **PROC FORMAT** to build the format, this will be used to indicate that a customer number is on the list supplied by the input file.

Line 5. This line places the value "onllist" into the variable called FMTNAME, which is used by PROC FORMAT to build the format, the value, can be any character string that does not end in a number and is seven characters or less.

Line 6. This statement writes the record to the temporary output file.

Line 7. This line tests for the end of the input file, if the variable EOF is set to a one (1) then the last record has been read **and** processed. This allows the program to output one last record to be able to have a format condition for all records not on the initial list of customers from the input file.

Line 8. This line places the value 'other' into the variable called START, which is used by **PROC FORMAT** to build the format. The output variable START in this case is a character variable, be aware that **PROC FORMAT** allows a special case of "other" as a default for any value not on the list, so the variable START is defined here as more than 5 characters.

Line 9. This line places the value "N" into the variable called LABEL, which is used by **PROC FORMAT** to build the format, this will be used to indicate that a customer number is not on the list supplied by the input file.

Line 10. This statement writes the last record to the temporary output file.

Line 13. This call to **PROC FORMAT** with the option **CNTLIN=** builds the user format, (in this case called "onlist") from the temporary file called fmt.

Line 16. This begins the **PROC SORT** command and defines the name of the input file.

Line 17. This line contains the "**WHERE=**" clause and is the heart of this example. This one statement selects all of the records in the input file that are also in the list file from line 2 above. The test is preformed before the record is presented to the sort routine for processing. This allows the programmer to skip the step of creating a full file of only the needed records before sorting that file, thus saving resources and time. The author has even seen code that sorts the big file and then merges it with a sorted list of customer numbers.
The **PUT=** command is a SAS Function that can be used to convert one value to another. The creative use of the **LABEL** variable allows a simple test to be set up so that either the true condition of ("Y" = "Y") or the false condition of ("N" = "Y") is created as the operand of the **WHERE=** test condition (where-expression). The binary output from this test selects or rejects the current record in the source file.

Line 18. This line of code defines the new output file. Failure to use this PROC SORT option will corrupt the original input file if all records are not selected.

Line 19. This BY statement defines the sort order of the output records.

Line 21. This line is optional but as the last line of a PC SAS job it is required to get the job finished. The **RUN** command also will write the SAS notes before the start of the next code step.

EXAMPLE 4 –USE OF COMPLEX FUNCTIONS IN A WHERE-EXPRESSION

This example will only be presented in the conference presentation if time permits, and shows the extreme complexity that is allowed in the "**where-expression**". Functions and macro variables can be used as an operand in the construction of a "**where-expression**". This is possible because the thrust of a "**where-expression**" is to create a binary result. This result chooses the records to be selected or rejected at run-time. Analysis of the output will be left as a student exercise and presented at the presentation.

TEST DATA FILE

Code the following data step and create a SAS Data file but choose only one %let statement from this list of two.

```
%let system = 49,49;   * if using an ASCII based computer (UNIX or a PC);
*let system = 241,241; * if using an EBCDIC based computer (IBM Mainframe);

Data TestText;
    A = 04; B = "1"; C = "0"; D = 'This author                ';output;
    A = 03; B = "1"; C = "1"; D = 'This frog                  ';output;
    A = 11; B = "1"; C = "0"; D = 'broke into 17              ';output;
    A = 06; B = "1"; C = "0"; D = 'has over 35 years          ';output;
    A = 13; B = "0"; C = "1"; D = 'dead logs.                 ';output;
    A = 02; B = "1"; C = "0"; D = 'This rock                  ';output;
    A = 15; B = "1"; C = "0"; D = 'of programming experience. ';output;
    A = 08; B = "0"; C = "0"; D = 'This prince                ';output;
    A = 09; B = "0"; C = "1"; D = 'jumped over 4              ';output;
    A = 01; B = "1"; C = "0"; D = 'kissed over 1000           ';output;
    A = 10; B = "1"; C = "0"; D = 'small pieces.              ';output;
    A = 12; B = "0"; C = "1"; D = 'sleeping ladies.           ';output;
Run;
Proc sort data=TestText (where=(
                        (substr(put(A,binary4.),2,1) = collate(&system)) and
                        (B ne C) and
                        (B = collate(&system))
                        ))
           Out=temp;
           By A;
Run;
Proc Print;
Run;
```

CONCLUSION

Each of these examples have shown some new way to mystify your friends and make SAS work harder and faster for you. Once these techniques are mastered and new things discovered it will be easy to eliminate unneeded steps from

your SAS code jobs. The biggest reason that most SAS jobs "Run So Long" and "Cost So Much" (as the IT guys say) is because the jobs have extra steps that can be eliminated by some of these techniques or doing more in each data step.

CONTACT INFORMATION

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

Name	William E Benjamin Jr
Address	P.O. Box 42434
City, State ZIP	Phoenix AZ, 85080
Work Phone:	602-942-0370
Fax	602-942-3204
E-mail:	wmebenjaminjr3@juno.com
	William@owlfunding.com