

Flexible Code Using Arrays With A Variable Name Wildcard

Jonathan Goddard, Independent Contractor, Southampton, UK

ABSTRACT

This paper illustrates two sparsely-documented features of Base SAS® which may be a useful addition to the programmer's toolbox: the variable name wildcard and the implicitly subscripted array. These can be combined to perform repetitive processing without specifying the number of variables, providing flexible code.

INTRODUCTION

Two Base SAS concepts that can be combined in a useful way are the variable name wildcard and the implicitly subscripted array. We shall look at each in turn.

VARIABLE NAME WILDCARD

In a data step, we can refer to lists of similar variables without having to write them out in full. This is normally done as follows, e.g.:

```
data d1 (keep = subject visit1-visit5);
```

However, it is possible to make this more general by using a variable name wildcard, a colon (:), e.g.:

```
data d1 (keep = subject visit:);
```

This will keep all variables beginning with VISIT.

IMPLICITLY SUBSCRIPTED ARRAYS

In a data step, we can define numeric and character arrays by writing:

```
array _narray <numeric variable-list>;  
array _carray $ <character variable-list>;
```

Notice that there is no array subscript {} here, which means we cannot refer to a particular element of the array. This is not necessarily important when all the elements are to be processed in the same way. The number of elements is instead defined by the number of variables in the variable list. Hence, this type of array is called 'implicitly subscripted'. By combining these two concepts, we can declare arrays in a flexible way, e.g.:

```
array _visit visit;;
```

This means that the elements of the _VISIT array are all numeric variables beginning with VISIT (e.g. VISIT1, VISIT2, VISIT3), in the order in which they occur in the data set. This will be important in certain scenarios, as we will see later.

```
array _trt $ trt;;
```

Likewise, the _TRT array contains all character variables beginning with TRT (e.g. TRT1, TRT2).

EXAMPLES

1. REPLACING BLANKS WITH ZEROS IN LABORATORY SHIFT TABLES

Suppose we have data sets containing shift table frequencies for particular parameters, e.g.:

PhUSE 2008

Haemoglobin (Low / Normal / High):

BASELINE	END_LOW	END_NORM	END_HIGH
Low	.	3	.
Normal	1	7	.
High	.	1	1

Urine Glucose (Negative / Trace / + / ++ / +++):

BASELINE	END_NEG	END_TRACE	END_POS	END_POS2	END_POS3
Negative	5	1	.	.	.
Trace	1	1	.	.	.
+	.	.	1	.	.
++	.	.	.	1	.
+++	1

Prior to reporting, we want to replace the missing values with zeros. This can be accomplished in a flexible way by looping over an implicitly subscripted array:

```
array _end end;;  
do over _end;  
    if _end = . then _end = 0;  
end;
```

Notice that the code is the same for both examples; we did not have to specify the number of elements in the array. Alternatives would involve hard-coding the names and numbers of elements, so this makes the above approach very flexible.

2. ADVERSE EVENT FREQUENCY TABLE WITH SUBJECT DETAILS

Suppose an adverse event table shell required details of the subjects experiencing each event, i.e.:

```
Preferred Term      n      %      Subject Details:  
  
xxxxxxxxxxxxxxxxxx  xx      xxx      102, 104, 106, xxx,  
                    xxx, xxx, xxx, ...
```

Consider a small example data set called EVENTS:

AEPREF	SUBJECT
Headache	102
Headache	104
Headache	106
Pain	101
Pain	102
Pain	107

To get all the subject numbers into a single variable, one way would be to transpose the subject numbers and use the variable name wildcard technique:

```
proc sort data=events;  
    by aepref subject;  
run;  
  
* Transpose the subject numbers onto a single row for each event;  
  
proc transpose data=events out=patients1 prefix=subj;  
    by aepref;  
    var subject;  
    id subject;  
run;
```

PhUSE 2008

The output data set PATIENTS1 contains:

AEPREF	SUBJ102	SUBJ104	SUBJ106	SUBJ101	SUBJ107
Headache	102	104	106	.	.
Pain	102	.	.	101	107

We want to concatenate these subject numbers in subject order, so this illustrates a potential pitfall of using a variable name wildcard: the order is determined by the order of variables in the data set. We need the transposed variables in subject number order, so we need another transposed data set to get this.

```
* Get transposed variables in subject order;

proc sort data=events out=order (keep=subject) nodupkey;
  by subject;
run;

proc transpose data=order out=order1 prefix=subj;
  var subject;
  id subject;
run;
```

The data set ORDER1 contains variables in the desired order:

SUBJ101	SUBJ102	SUBJ104	SUBJ106	SUBJ107
----------------	----------------	----------------	----------------	----------------

We are not interested in the data; only the order of the variables. We will read this order (but no data) into the next data step:

```
options missing=''; * Prepare conversion to missing character;

data patients2;
  set order1 (where = (0)) patients1; * No data from ORDER1;
  length allsubj $2000;
  array _subj subj;;
  do over _subj;
    if allsubj = '' then allsubj = compress(put(_subj,8.));
    else if _subj ne . then
      allsubj = trim(allsubj) || ', ' || compress(put(_subj,8.));
  end;
run;
```

The data set PATIENTS2 has a variable ALLSUBJ, containing all subjects experiencing each event:

AEPREF	ALLSUBJ
Headache	102, 104, 106
Pain	101, 102, 107

Notice that the code does not specify the number of elements in the array, making it generic and re-usable.

PhUSE 2008

CONCLUSION

The solutions given are not the only way to solve these problems (Example 2 could be solved using a data step with RETAIN, for example), but I like the simplicity of the process described here. It will be a useful option for a SAS programmer to have.

Other scenarios where this technique could be employed are:

- 1) Post-processing formatting changes for a variable number of columns (usually treatment groups) immediately prior to PROC REPORT (similar to Example 1).
- 2) Simple calculations over timepoints: for example, identifying subjects with consecutive missing values. This would be similar to Example 2, as we would need the timepoints in the correct order in the transposed data set. (Also, we would need to handle any repeats within a timepoint in an appropriate way).
- 3) Conversion of various types of laboratory results from character to numeric (e.g. results in standard units, results in local units, results in protocol units, etc.)

CONTACT INFORMATION

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

Jonathan Goddard
Independent Contractor
Email: jonathan.goddard@dsl.pipex.com

Brand and product names are trademarks of their respective companies.