Using Lists To Replace Formats In SCL Applications

Don Stanley
Information Power
Wellington
NEW ZEALAND

OVERVIEW

This paper shows how some SCL applications that use SAS user defined formats can be enhanced by changing from using formats to lists.

Often, we use formats in an SCL application simply because that is what we used in SAS data and proc step coding. We know and understand user defined formats from our base SAS use. However, in SCL applications, formats may not be appropriate for optimal performance.

REVIEW OF FORMATS

(1) Little control over contents from SCL

In order to alter the format you must submit code

(2) No control over removal from memory

Once a format is loaded it is up to SAS internal mechanisms as to when the format is removed

(3) No ability to add to without re-building entire format

You may load a very large format, then need to add an item (perhaps by allowing a user to add to an underlying dataset from which the format is sourced). However, formats force you to re-create the whole format, which can be an enormous amount of work to add a single item

(4) No ability to mix character and numeric

Formats are either strictly numeric or character. There are times when the ability to use either without loading multiple formats is useful.

(5) Strictly oriented to changing the look of a variable, or remapping a variables value into another
(6) Can only match on whole words in the value side of the format

A format requires to make a match on an entire word. There is no flexibility for trying to attain a match on, say the first three letters of a character value.

**WHEN CAN WE USE LISTS INSTEAD OF FORMATS**

(1) *Replacing Formats With Lists When Used As Lookup tables*

A lookup table is a list of data items that are used to verify some data entry is correct. For example, I may need to enter a client identifier into a frame text field. I could accept the field entry, then use the format to verify validity of the entry. The format may look as follows:

```sas
proc format;
  value $isvalid 'X00000', 'Y00000', 'Z00000' = 'Y' other = 'N';
run;
```

The associated SCL code may look like

```sas
if put(text_field,$isvalid.) eq 'N' then do;
  error processing;
  return;
end;
```

Using this approach, I strike the following issues

- the format cannot be unloaded once loaded, so uses memory even when finished with
- in order to add any items to the format submitted code must be run to completely rebuild the format. This may in fact be difficult, due to the nature of the application (possibly in FSEDIT or called by PROC DISPLAY), through to impossible (the format has already been loaded and couldn't use new items anyway)

Using a list I can overcome these issues as follows

- I can use the list to check the entered value, then destroy the list and return the memory
- I can add an item to the list at any stage, in SCL code.

The code to use the list follows. Note that here I have created the list, but of course the list may be sourced from an SLIST entry, a catalog, a dataset or other valid list sources.

```sas
listid = makelist();
rc = insertc(listid,'X00000');
rc = insertc(listid,'Y00000');
rc = insertc(listid,'Z00000');
```
if not searchc(listid,text_field) then do;
  .. error processing;
  return;
end;

(2) Converting variable values

This is perhaps the most understood and well used part of SAS format processing. Here we take a value and create a new variable (or modify the current variable's value) by using the format.

The following code is a sample INIT section to illustrate how to implement lists rather than formats for this task.

```
length label format $ 15 start test $ 5;
init:
  submit continue;
  proc format;
    value $test
       'BCRD' = 'Bankcard'
       'MCRD' = 'Mastercard'
       'MGCD' = 'Mastercard Gold'
       'VISA' = 'VISA'
       other = 'N/A';
  run;

data cards;
  test = 'BCRD';
  output;
  test = 'MCRD';
  output;
  test = 'MGCD';
  output;
  test = 'VISA';
  output;
  test = 'TEST';
  output;
run;
endsubmit;
```

dsid = open('work.cards','i');
call set(dsid);
rc = time(); put 'START FORMAT' rc=;
do while (1);

104
rc = fetch(dsid) ;
if rc eq -1 then leave ;
format = put(test,$test.) ;
put test= format= ;
end ;
rc = time() ; put 'END FORMAT ' rc= ;

listid = makelist() ;
rc = insertc(listid,'Bankcard',-l,'BCRD') ;
rc = insertc(listid,'Mastercard',-l,'MCRD') ;
rc = insertc(listid,'Mastercard Gold',-1,'MGCD') ;
rc = insertc(listid,'VISA',-l,'VISA') ;
rc = insertc(listid,'N/A',-l,'OTHER') ;
rc = time() ; put 'START LIST ' rc= ;
rc = rewind(dsid) ;
do while (1) ;
  rc = fetch(dsid) ;
  if rc eq -1 then leave ;
  if nameditem(listid,test) then
    format=qetitemc(listid,nameditem(listid,test)) ;
  else format = qetnitemc(listid,'OTHER') ;
  put test= format= ;
end ;
rc = time() ; put 'END LIST ' rc= ;
call close(dsid) ;
return ;

In the above code, a number of PUT statements are placed to give an indication of the difference in execution time between the FORMAT approach and the LIST approach. This is the output that the above produces:

START FORMAT RC=37287.54
TEST=BCRD FORMAT=Bankcard
TEST=MCRD FORMAT=Mastercard
TEST=MGCD FORMAT=Mastercard Gold
TEST=VISA FORMAT=VISA
TEST=TEST FORMAT=N/A
END FORMAT RC=37287.58999999999
START LIST RC=37287.65
TEST=BCRD FORMAT=Bankcard
TEST=MCRD FORMAT=Mastercard
TEST=MGCD FORMAT=Mastercard Gold
TEST=VISA FORMAT=VISA
TEST=TEST FORMAT=N/A
END LIST RC=37287.65

You can see that the list approach is quicker even for this small dataset. The approximately 5% difference is likely to be more significant as format sizes increase.

(3) Are there situations where lists should not be used instead of formats?

It has been suggested to me that very large formats should not be replaced with lists. I do not agree with this, as the essential disadvantage of formats, the inability to remove them from memory once loaded, is even more of an issue.

Formats carry out a binary search to find any value that needs converting to a label. I am not aware of just what algorithm the SEARCHC function uses. Possibly formats are faster, possibly not. It has been my experience that there is little perceivable difference between the two, but the time to load a format may be greater.

Of course, some situations exist where formats must be used. An example is if you wish to output a field from SCL using one of SAS Institute’s formats. In that case formats must be used (this covers the case where the Institute format is embedded in a user format also).

Also, if your format uses special format attributes such as PICTURES, HLO, RANGES, _SAME_ and various other attributes, you are probably best off maintaining the format in preference to the list.

(4) Converting a format to a list

You can do this using the CNTLOUT option of PROC FORMAT and using the resulting dataset to fill the list. The SCL code is:

```sas
/* convert a format to a list */
submit continue;
   proc format library=work cntlout=work.temp ; run;
endsubmit;

dsid01 = open('work.temp(keep=start label)','i');
call set(dsid01);

listid2 = make1ist();
do while(1);
   rc = fetch(dsid01);
   if rc eq -1 then leave;
   if index(start,'OTH') then
```
rc = insertc(listid2, label, -1, 'OTHER') ;
else rc = insertc(listid2, label, -1, start) ;
end ;
call close(dsid01) ;

What this allows is a format to be the underlying data structure that a list is based on. This may be very advantageous if the format is also used in batch processing where the list is less simple to use. What I have done here is shown a way to store the optimal structures for BASE SAS and SCL programming.

**SUMMARY**

The above section shows how dynamic lists can replace less dynamic format tables in SCL applications. The advantages of this approach are reduction in memory used and more flexible, more responsive applications.

Examples above have built the lists in the code. In general, you will most likely want to create the list once (or as an underlying format table changes in (4) above), and save the list in an SLIST entry. This will result in less cluttered code and less time used at runtime.

The author can be contacted at email address DSTANLE@IBM.NET, or via Box 10 603, Wellington, New Zealand.

**REFERENCES**

SAS Screen Control Reference, Version 6, Second Edition