Harnessing the Power of SCL Lists

Lisa Ann Horwitz, SAS Institute Inc., New York

Abstract

SCL lists are ordered collections of data stored in memory, and are a powerful tool for applications developers. The sources of this data can include hard coded values, SAS data sets, information retrieved from interaction with the application user or the current SAS session, or the FRAME entry's objects themselves. This paper explores how to create and populate SCL lists, how to retrieve and utilize information from lists, how to manipulate lists, and how to access some of the lists associated with FRAME entry objects. An assortment of ideas for using SCL lists will be suggested, as well.

Introduction

When building a FRAME application, there is often a need for storing and passing information from one part of the SCL program to another or from one SCL program to another, and for extracting information from a SAS data set for the purpose of performing validation or populating a selection list. Conveniently, a structure exists that not only facilitates these tasks, but is useful in many other contexts. This structure is an SCL list. This paper describes what an SCL list is, how it can be created and manipulated, and offers some applications utilizing SCL lists that just begin to demonstrate their power and usefulness.

The FRAME applications built to support this paper were developed under Version 6.12 of the SAS® System on the Windows 95/Windows NT platform. Many dramatic and wide ranging changes are in store for SAS/AF® and the rest of the SAS System. However, all Version 6 applications will work in Version 7 with no changes. Also, SCL lists continue to be a powerful application developer's tool as well as part of the underpinning of the application classes themselves. Two of the new SCL list functions that address V7 case sensitivity support are listed at the end of the paper.

What is an SCL List?

An SCL list is an ordered collection of data. It is stored in memory, making access to its information very fast. It is doubly linked, which means that the information stored within a list can be accessed either from the front of the list or from the back of the list. An SCL list is available for use anytime in an SCL program. These characteristics might remind some programmers of arrays, and, indeed, both are used to store and retrieve information.

However, SCL lists are far more versatile and powerful than arrays for the following reasons. Consider a typical array definition, as found in a standard data step or SCL program:

```
array testlst {4} $ 3;
```

This statement creates an array named *testlst*, which contains four elements, each of which is of type character, and with a length of three bytes. The elements are named, by default, testlst1, testlst2, testlst3, and testlst4.

This statement demonstrates the characteristics of arrays. Arrays have a fixed number of elements, and each element has a predetermined length, which is the same for all the elements. The elements must all be of the same type - character or numeric. The elements are referred to by their index number. Arrays cannot be stored permanently.

Compare these characteristics to those of an SCL list:

- An SCL list does not need a predetermined number of elements. Whether a list is created with any elements or none, the length of the list can grow dynamically.
- The elements in an SCL list do not need a predetermined length, and each element can have a different length than the other elements - and these lengths can also change dynamically as the list is used.
- The elements do not have to have a predetermined type, and the elements do not

have to have the same type: the elements can contain character or numeric values, or list ID's (the identifiers to other lists - this will be discussed in greater detail below).

- The elements in an SCL list can be referred to by their index number or by a name assigned to the element. Thus, a "slot" in an SCL list can contain three pieces of useful information: its position in the list (index), its name, and its value.

- An SCL list can be stored permanently, either as a file or as a catalog entry, thereby enabling access to information across application invocations.

How are SCL Lists Created and Manipulated?

SCL lists are created and manipulated through the use of SCL functions. There are SCL list functions to

- create and delete lists
- populate and depopulate lists
- read from and write to SCL lists
- sort lists
- create a pulldown window from list contents
- print a list
- store a list permanently.

Many SCL list functions have the following basic syntax:

```
variable=
   SCL-LIST-FUNCTION(argument1,
   argument2,argumentn);
```

As regards this syntax, consider the following rules of thumb:

- If the function name starts with GET, the function is used to read or retrieve a value from the SCL list. If the function name starts with SET, it is used to assign or insert a value into the list. For example, compare GETITEMN and SETITEMN: the first is used to retrieve a value from the list, and the second is used to insert an item into the list.

- If the function has the letter "N" in the middle of its name, the function refers to a named element in the list. Otherwise, the function refers to an indexed element in the list. For example, compare functions GETNITEMN and GETITEMN: the first refers to a named element, and the second, to an indexed element.

- If the function has an L at the end of the name, the function performs its task on an element containing a list id. If the function has a C at the end of the name, the function performs its task on a character element. Similarly, an N dictates that the function performs its task on a numeric element. For example, consider GETITEML, GETITEMC, and GETITEMN: the first retrieves a list id, the second retrieves a character value, and the third retrieves a numeric value. This point highlights the fact that though a list can contain all three types of elements, it is necessary to know what sort of value the SCL list "slot" contains. The ITEMTYPE function returns the type of an element, if this information isn't known.

Functions following these patterns are demonstrated in the sample programs shown later in the paper.

The MAKELIST function is used to create a new SCL list. Consider these two examples:

```
newlist1=makelist();
newlist2=makelist(5);
```

The first statement creates a null list, a list with no elements. The second creates a list with five elements. In either case, as mentioned earlier, the lists can grow and shrink to accommodate elements as necessary. NEWLIST and NEWLIST1 are the names of the lists being created. The lists will always be referred to by these names. More specifically, however, these names are SCL variables whose values are the numeric identifiers of the lists. Although we can view the values of these numeric variables, as programmers we don't really need to be concerned with what the values are, as long as we refer to the lists by these names - their list id's.

Once a list is no longer needed, it can (and should) be deleted from memory by using the DELLIST function:

```
rc=DELLIST(newlist);
rc=DELLIST(newlist1);
```

The use of a variable named “rc” (short for “return code”) is by convention. Its value can be monitored to be sure that the function has performed its task properly.

You can also clear a list, in other words retain its structure but remove the values from the elements, by using the CLEARLIST function:

```
rc=CLEARLIST(newlist);
rc=CLEARLIST(newlist1);
```

Once a list has been created, it is typically populated with values needed in the application program. There are several ways to populate a list. One is to use the INSERTC function. Consider the following segment of SCL code:

```
❶ colorlst=makelist();
❷ colorlst=insertc(colorlst,
   'red',1);
❸ colorlst=insertc(colorlst,
   ‘white’,2);
❹ colorlst=insertc(colorlst,
   ‘blue’,3);
```

This code ❶ creates an SCL list named *colorlst*, and then inserts into the ❷ first, ❸ second, and ❹ third elements of the list the character values “red,” “white,” and “blue,” respectively. The repetition of the name of the list at the left of the equal sign and as the first argument of the INSERTC function implies that the elements are being loaded into the existing list.

It is handy to be able to see the contents of the list during the development of the application to insure that the list is being populated with the correct values. A routine that enables us to perform this task is PUTLIST:

```
CALL PUTLIST(colorlst,
   "COLOR LIST",0);
```

The first argument of the routine is the name of the list to be inspected; the second (optional) argument is a heading to be used in the display of the contents; and the third (optional) argument indicates whether to indent from the margin. A “0” in this position means to display the contents vertically rather than horizontally, in paragraph form. This usually makes for easier interpretation of the contents, and is especially convenient when a list contains sublists.

The preceding call to the PUTLIST routine displays the following in the SCL Log:

```
❶ COLOR LIST(❷ ‘red’
             ❸ ‘white’
             ❹ ‘blue’
            ) ❺[1903]
```

This display shows ❶ the label (“COLOR LIST”), the three character elements (❷ “red,” ❸ “white,” and ❹ “blue”), and ❺ the list id (1903). And, due to specifying the “0” as the indent factor, the elements are listed vertically.

It is also possible to populate the list by assigning the indices from the rear of the list, indicated by the index value of -1. The values of the elements and their order are identical to those in the previous example.

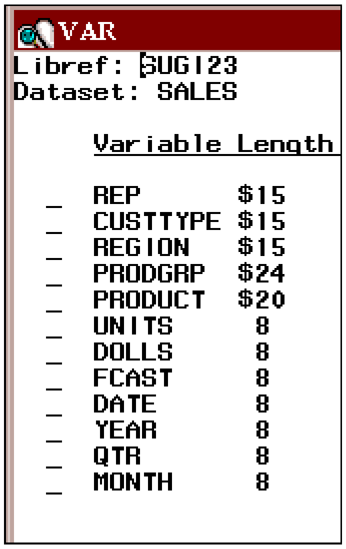
```
colorlst2=makelist();
colorlst2=insertc(colorlst2,
   ‘red’,-1);
colorlst2=insertc(colorlst2,
   ‘white’,-1);
colorlst2=insertc(colorlst2,
   ‘blue’,-1);
```

Just as it is possible to populate a list with hardcoded values in several ways, it is also possible to retrieve the values in several ways. One way is to use the GETITEM family of functions. Consider this example:

```
❶length color1 color2 color3 $ 5;
❷ color1=getitemc(colorlst2,1);
❸ color2=getitemc(colorlst2,2);
❹ color3=getitemc(colorlst2,3);
```

❶ The LENGTH statement is used to assign lengths of five bytes to each character variable; the default length would have been 200 bytes. The three calls to the function *getitemc* assigns ❷ the first character item in the SCL list *colorlst2* to *color1*, ❸ the second, to *color2*, and ❹ the third, to *color3*.

Another way to populate an SCL list is to use the contents of a SAS data set. For example, a sample data set containing sales information for a computer software company is named SUGI23.SALES. Below is a portion of a screen capture of the VARIABLES window showing the variables in the SUGI23.SALES data set.



```
Libref: SUGI23
Dataset: SALES

     Variable Length

 _   REP       $15
 _   CUSTTYPE  $15
 _   REGION    $15
 _   PRODGRP   $24
 _   PRODUCT   $20
 _   UNITS       8
 _   DOLLS       8
 _   FCAST       8
 _   DATE        8
 _   YEAR        8
 _   QTR         8
 _   MONTH       8
```

To populate an SCL list with the unique values of the variable REP (sales representatives), use the LVARLEVEL function:

```
❶ salesid=open('sugi23.sales');
❷ nlevels=0;
❸ replist=makelist();
❹ rc=lvarlevel(salesid,'rep',
   nlevels,replist);
```

This code ❶ opens the SAS data set using the OPEN function, ❷ assigns variable *nlevels* the default value of zero, ❸ creates an SCL list named *replist*, and ❹ uses the lvarlevel function on the open data set (identified with the data set id *salesid*), identifying *rep* as the variable whose unique values should be used to populate the SCL list *replist*. After the fourth statement executes, the variable *nlevels* contains the number of unique values of the variable *rep*.

To see the contents of the list, again use the PUTLIST function:

```
call putlist(replist,'List of
   Reps',0);
```

The display looks like this:

```
List of Reps ('Lisa           '
              'Chuck          '
              'Nancy          '
              'Mark           '
              'John           '
              ) [1903]
```

In an application, it could be useful to display such information in a popup list for the purpose of user selection. This technique is especially handy because such a popup list does not take up permanent real estate on the window. The POPMENU function is used to populate and display a popup list:

```
whichcol=popmenu(colorlst);
whichrep=popmenu(replist);
```

The first call to POPMENU displays the list of colors *red*, *white*, and *blue*; the second displays the names of the sales reps:

```
red
white
blue
```

```
Lisa
Chuck
Nancy
Mark
John
```

The POPMENU function returns the number of the selected row.

Elements in SCL lists can be the list id's of other lists. Thus, lists are multidimensional in nature. For example, consider this example. First, create a list of demographic information pertaining to one of the sales reps in the SUGI23.SALES data set.

```
demoglist=makelist();
❶demoglist=insertc(demoglist,
   '325 E. 78th Street',1,
   'ADDRESS');
❷demoglist=insertc(demoglist,
   ‘New York’,2,’CITY’);
❸demoglist=insertc(demoglist,
   ’NY’,3,’STATE’);
```

The SCL list *demoglist* consists of three named elements: ❶ ADDRESS, ❷ CITY, and ❸ STATE.

Using the PUTLIST routine shows:

```
(ADDRESS=’325 E. 78th Street’
 CITY=’New York’
 STATE=’NY’ )[1903]
```

Next, create an SCL list of human resources information about the same sales rep:

```
hrlist=makelist();
hrlist=insertc(hrlist,
  ‘Associate Sales Rep’,1,’TITLE’);
hrlist=insertn(hrlist,55000,2,
   ‘SALARY’);
```

Using the PUTLIST routine shows:

```
(TITLE=’Associate Sales Rep’
 SALARY=55000 )[1903]
```

Next, create a list that contains the other lists.

```
replist=makelist();
replist=insertc(replist,’John’,1,
   ‘NAME’);
replist=insertl(replist,
   demoglist,2,’DEMOGRAPHICS’);
replist=insertl(replist,hrlist,
   3,’PERSONNEL INFO’);
```

Using PUTLIST shows:

```
(NAME=’John’
 DEMOGRAPHICS=
    (ADDRESS=’325 E. 78th Street’
     CITY=’New York’
     STATE=’NY’)[1903]
 PERSONNEL INFO=
    (TITLE=’Associate Sales Rep’
     SALARY=55000)[1905]
```

```
)[1907]
```

The sequence of events to retrieve an item from a sublist is to first find the list id of the sublist in which the item is located, and then retrieve the item. So, to retrieve salary, retrieve the list id for the sublist named PERSONNEL INFO:

```
hrsub=getniteml(replist,
   ‘personnel info’);
```

Then, read the value of the named element *salary*:

```
salary=getnitemn(hrsub,’salary’);
```

Interestingly enough, names used to identify elements in a list can be re-used. This means that these same three names, NAME, DEMOGRAPHICS, and PERSONNEL INFO, can be used repeatedly to identify elements containing information about the other sales reps. Each repeating pattern can be identified by occurrence number. Thus, the first occurrence of an named element NAME refers to the name of the first rep, the second occurrence of the named element NAME corresponds to the name of the second sales rep, and so on.

SCL Lists and FRAME Objects

An additional and extremely useful fact about SCL lists is that they are utilized by the objects in FRAME entries to store information about themselves and the application user’s interaction with them. Specifically, SCL lists store the values of the object’s instance variables. These values can be accessed by calling methods which return the list id’s of the populated lists, and can be very helpful in controlling the behavior of the application.

For example, the _GET_VALUE_ method provides the list id of the SCL list populated by interacting with a graphics object. Similarly, the _GET_INFO_ method provides the list id of the SCL list populated by interacting with hotspots in a SAS/GRAPH® output object. The syntax of these methods follows:

```
call notify(objectname,
   '_GET_VALUE_',SCL-list-name);
```

```
call notify(objectname,
   '_GET_INFO_',SCL-list-name);
```

Consider an application that displays the total sales for the five sales reps in a bar chart generated by a graphics object. Clicking on a bar changes the display to show another bar chart with that sales rep's sales for the previous year's four quarters. This behavior is easy to orchestrate by calling the _GET_VALUE_ method. Using PUTLIST to see the list that is created shows:

```
( DEPVALUE=( VALUE=69903150
            )[1967]
  INDTYPE='C'
  TEXT=''
  DEPTYPE='VBAR'
  INDVALUE='Mark'
  ID=''
  GROUP=''
 )[1965]
```

The named element INDVALUE indicates that the bar for the sales rep MARK was selected, and VALUE contains the total sales represented by the bar. This information can be used to subset the data set by rep="MARK" and to assign a title which includes Mark's total sales.

Next, consider a SAS/GRAPH-generated map of the United States with the four sales regions hotspotted. Using the _GET_INFO_ method allows easy identification of which hot spot was clicked on and makes further processing straightforward. The portion of the list containing information after a hotspot click event looks like this:

```
( SPOT=( NAME='ROCKYMT'
         LABEL='ROCKYMT'
         USERATTR=()[49]
         SPOTID=2
         HILITE=1
         HCOLOR='CYAN'
```

The hotspot named ROCKYMT was selected; this information can be used to further subset the data set to reveal the detail information for sales in that region.

The presentation of this paper includes a demonstration of these FRAME applications and several others which make use of SCL lists in various ways. The SCL code for these applications is included at the end of this paper.

Version 7: A Quick Word

SCL lists continue to be a powerful developer's tool and the means of storage of instance variables in Version 7. The elements available in the lists associated with Version 7 components will enable developers to monitor and trigger events, as illustrated in this paper.

One change that is far-reaching in Version 7 is mixed case support. Accordingly, functions have been added to SCL (in Version 7, "SAS Component Language" rather than "Screen Control Language") to allow for either Version 6 compatability (the default) or Version 7 case sensitivity. For example, the function SETLATTR can be used to set case sensitivity for an SCL list:

```
rc=setlattr(listid,'HONORCASE');
```

The default behavior is to 'IGNORECASE'.

Similarly, the NAMEDITEM function, which searches an SCL list for the occurrence of a named item, has a new parameter to specify whether case sensitivity should be honored in the name:

```
index=nameditem(listid,name,
   <occurrence,<startindex,
   <force-UP>>>);
```

Applications using SCL Lists: Sample Code

Application 1. Using SCL Lists to Display POPMENUS and to Display a Subset of a SAS Data Set in a Datatable. FRAME includes pushbuttons to select category to subset by: sales rep, product, product group, or customer type.

```
/****************************************/
/* Use the LVARLEVEL function to       */
/* populate an SCL list. Use the       */
/* POPMENU function to display the     */
/* values in the SCL List in a popup   */
/* menu. Use an SCL List to subset a   */
/* Data Table.                         */
/***************************************/

length clause $ 30 name $ 8;
rc=rc;

init:
   salesid=open('sugi23.sales');
   infolist=makelist();
   wherlist=makelist();
return;

main:
   call notify('.',
     '_get_current_name_',name);
   nlevels=0;
   rc=lvarlevel(salesid,
      name,nlevels,infolist);
   whichrow=popmenu(infolist);
   value=getitemc(infolist,whichrow);
   clause=name||'='||quote(value);
   call notify('title',
     '_set_text_',
     'Subset by '||clause);
   wherlist=insertc(wherlist,clause,1);
   call notify('table',
      '_set_where_',wherlist);
   rc=clearlist(wherlist);
return;

term:
   rc=dellist(infolist);
   rc=dellist(wherlist);
   rc=close(salesid);
return;
```

Application 2. Using SCL Lists to Populate a List Box and to Provide Information for Use in PROC TABULATE Code.

```
/***************************************/
/* populate SCL list VARLIST with      */
/* labels of variables, and name the   */
/* SCL elements with the names of the  */
/* variables. Use the labels to fill a */
/* List Box, and use the names to run  */
/* PROC TABULATE code.                 */
/***************************************/

length text $ 14 name $ 10;
rc=rc;
name=name;

init:
   varlist=makelist();
   varlist=insertc(varlist,
     'Product Group',1);
   varlist=insertc(varlist,
     'Customer Type',2);
   varlist=insertc(varlist,'Product',3);
   varlist=insertc(varlist,'Region',4);
   varlist=insertc(varlist,
     'Sales Rep',5);
   name=nameitem(varlist,1,'prodgrp');
   name=nameitem(varlist,2,'custtype');
   name=nameitem(varlist,3,'product');
   name=nameitem(varlist,4,'region');
   name=nameitem(varlist,5,'rep');
return;

variable:
   call notify('variable',
      '_get_last_sel_',row,issel,text);
   classvar=nameitem(varlist,row);
   submit continue;
      options ps=60 ls=132 nocenter
         nodate nonumber;
      proc tabulate data=sugi23.sales
         format=dollar14.;
         class &classvar month;
         var dolls fcast;
         tables month,&classvar*dolls*sum
        / rts=14;
         keylabel sum=' ';
         label dolls='Total Sales';
         title
           'Sales for &text by Month';
      run;
   endsubmit;
   rc=woutput('clear');
return;


term:
   rc=dellist(varlist);
return;
```

Application 3. Using a Catalog Entry to Save the Contents of an SCL List. FRAME entry contains radio boxes to select options.

```
/***************************************/
/* This entry shows how to store values*/
/* from an SCL list to a catalog entry */
/* - OPTIONS.SLIST - and how to reload */
/* the contents from the catalog to    */
/* the list in order to reinitialize   */
/* screen variables.                   */
/**************************************/

rc=rc;

init:
   options=makelist();
   if exist('sugi23.demo.options.slist')
   then do;
      rc=fillist('catalog',
        'sugi23.demo.options.slist',
        options);
      call notify('date','_activate_',
         getitemn(options,1));
      call notify('center','_activate_',
         getitemn(options,2));
      call notify('printer','_activate_',
         getitemn(options,3));
   end;
return;

date:
   call notify('date','_is_active_',
      datenum);
return;

center:
   call notify('center','_is_active_',
      cennum);
return;

printer:
   call notify('printer','_is_active_',
      printnum);
return;

save:
   options=insertn(options,datenum,1);
   options=insertn(options,cennum,2);
   options=insertn(options,printnum,3);
   rc=savelist('catalog',
     'sugi23.demo.options.slist',
     options);
   call execcmd('end');
return;


term:
   rc=dellist(options);
return;
```

Application 4. Using SCL List Populate