

Simplified Selecting: A Comparison of SCL Selection List Techniques

Frank C. DiIorio, Trilogy Consulting Corporation

Introduction

One of the most dramatic trends in the computer industry in recent years has been the popular acceptance of the graphical user interface (GUI). GUIs and their character-based antecedents, when implemented correctly, simplify access to complex applications. SAS software has not overlooked this trend. The introduction of SAS/AF and Screen Control Language (SCL) in Version 6.06 gave application developers a host of powerful tools to craft interactive systems.

A common task performed in GUI-style applications is selection from a list of values. Rather than require the user to enter, say, the name of a company department, the application's screen displays appropriate choices. The advantages are numerous: the user does not have to commit codes to memory, specification of the code is quick, and the user is assured that by its very presence on the screen the choice is valid.

Selection from a list has several aspects. One is degree of change. The list may be static, and can be "hard-coded" in the application (American states, for example). At the other extreme, the list items may be dynamic, rapidly changing (e.g., customer names, adverse event codes in a clinical trial). Another list feature is the number of selections which may be made. In some cases a single choice is appropriate (e.g., country of birth). Two or more choices are appropriate in other cases, such as identification of a group of job titles.

This Paper

While SCL offers a variety of tools to build selection lists, it is not always obvious to the developer which is most appropriate for the task at hand. Tool selection can be particularly baffling to the novice AF developer.

This paper outlines some of the more popular mechanisms for developing selection lists and discusses their strengths and limitations. The techniques include: Choice Groups, the LISTx and DATALISTx functions, and the use of ex-

tended tables for selection lists.

The paper assumes some knowledge of SCL. Attention is focused on features available in Version 6.06 and higher of SAS software. The paper is *not* an in-depth tutorial: its role as an overview and comparison precludes extended discussions of syntax and useful but seldom-used options.

Notice that for the sake of brevity only the relevant portions of the SCL Display and Source entries are presented. Likewise, purely aesthetic features such as window sizing, color, and titling are mentioned only in passing. These features, while important to the overall effectiveness of an application, do not have a direct role in the programmer's selection of techniques.

Method 1: Hard-Coding

When choices are static (the States example given earlier), several "brute force" options are available. Two techniques are presented below, for single and multiple selections.

Single Selection. A simple and highly effective mechanism for restricting the user to a single selection is the Choice Group. Related fields are identified as a Group in the ATTR screen's Choice Group attribute. Selection is indicated by placing marking one of the fields with a non-blank character and pressing Enter, or by a mouse click on the desired field. Only one choice is allowed. If the user tries to specify a second choice, the first selection is cancelled. The choice is made known to the SCL program via the ISACTIVE function.

Suppose the user has a choice of three Departments. Three fields (D1, D2, and D3) are placed on the screen, along with some descriptive text. In the ATTR window each field is assigned to Choice Group DEPT.

Exhibit 1 presents one application of the Choice Group. SCL variable CHOICE is assigned on the basis of the selection. If ISACTIVE returns a 0, a message is displayed and execution of MAIN terminates.

Multiple Selections. The conversion of a forced,

Display

```

_Marketing      <- Field D1, Choice Grp DEPT
_Sales          <- Field D2, Choice Grp DEPT
_Publications   <- Field D3, Choice Grp DEPT

```

Source

```

main: x = isactive('dept');
      select (x);
      if x = 0 then do;
        _msg_ = 'Please specify';
        cursor d1;
        return;
      end;
      else if x = 1 then dcode='MKT';
      else if x = 2 then dcode='SAL';
      else if x = 3 then dcode='PUB';

```

Exhibit 1 Choice Groups

single selection list into a multiple choice display is achieved by simply removing the Choice Group designation in the ATTR screen. Rather than have the single selection neatly passed to the program via the ISACTIVE function, the programmer must evaluate each field individually. An example is presented in Exhibit 2.

DISPLAY

<same as Exhibit 1, but no Choice Group assignment>

SOURCE

```

main: if d1 ^= ' ' then mkt = 1;
      else mkt = 0;
      if d2 ^= ' ' then sal = 1;
      else sal = 0;
      if d3 ^= ' ' then pub = 1;
      else pub = 0;
      if mkt + sal + pub = 0 then
        _msg_ = 'WARNING: no choices made!';

```

Exhibit 2 Hard-coded Multiple Selections

Review: Advantages Hard-coded lists are easy to code in cases where list items change relatively infrequently. The values, descriptors, and the program manipulating them are in the same catalog entry, which makes program maintenance relatively straightforward.

Review: Drawbacks The binding of list to program noted above is also a drawback, since lists are not readily shared among PROGRAM entries. Changes to the list require modification of the DISPLAY, SOURCE, and (if Choice Groups are used) ATTR screens. These changes, in turn, necessitate recompiling the program.

Method 2: SHOWLIST Function

The SHOWLIST function presents a list of values, usually hard-coded, in a more visually appealing format. The user specifies a list of up to 13 values, along with a prompt message. When activated, SHOWLIST displays a pop-up window. The user scrolls to the desired entry or points to it with the mouse. Upon exit, the specified value is returned to a character value. This process is shown in Exhibit 3.

DISPLAY

Enter Department, or leave blank to review choices: ____ <- Field DEPT

SOURCE

```

main: if dept = ' ' then
      dept=showlist('Marketing',
                    'Sales',
                    'Publications',
                    'END to exit');
      else <validate DEPT>;

```

SHOWLIST Window

```

END to exit

Marketing
Sales
Publications

```

Exhibit 3 SHOWLIST Function

Review: Advantages SHOWLIST is easy to code and, since its parameters may be either character constants or variables, it has limited dynamic capabilities. Both data and program are in the same PROGRAM entry, making maintenance fairly straightforward. Finally, the pop-up window conveys a more professional feel.

Review: Drawbacks Only 13 options are permitted, and the SHOWLIST call must return a

single character value (of course, conversion to numeric may be possible in some circumstances). In the window itself, only a single field is displayed: annotation or explanation of values is impossible. The default color, window size and placement, and PMENU for SHOWLIST may not be well integrated with the rest of the application (although most of these features can be overridden).

Finally, since the SHOWLIST window itself cannot contain any underlying SCL code, it is impossible to tell whether the user actually selected an option until return controls to the calling SCL program. If a value *must* be chosen the SHOWLIST call has to be enclosed in a loop and re-displayed until a value is specified.

Method 3: LISTx Functions

The appealing "look and feel" of the SHOWLIST function was mitigated by its arguments being tightly bound to the program and by the small number of items which could be displayed. The LISTC and LISTN functions address these and other shortcomings.

The LISTx functions require the presence of another catalog entry, usually of type LIST (HELP and MENU types are also valid). The entry contains a list of numeric or character values. When invoked, the LISTx function displays the LIST, HELP, or MENU entry, and prompts the user to make a selection. Selection technique is identical to SHOWLIST: a mouse click or manual scrolling, followed by pressing the Enter key at the desired item.

Exhibit 4 illustrates the use of the LISTC function to return a character value to the field DEPT. LISTN would be used to return a numeric value. In the example, LISTC displays DEPTS.LIST in the current directory (a four-level name could be used to identify an entry in another catalog). Since no other parameters are entered, a default prompt is displayed and the window closes once a selection is made.

Review: Advantages Like SHOWLIST, the LISTx functions are easy to code and result in quick development of appealing displays. The separation of the list values from the program utilizing the list can be a significant benefit. If Department values need to be accessed by more than one application, the LIST entry may be

SOURCE

```
main: if modified(dept) | dept = ' ' then
      dept = listc('depts.list');
```

LISTC Pop-Up Window

Please make 1 selection

- Accounting
- Finance
- Human resources
- Marketing
- Promotions
- Travel

Exhibit 4 Using the LISTC function

shared. Maintenance and utility of the list are greatly increased.

Review: Drawbacks LISTx functions return only one value from a list. If the developer needed to prompt the user for a varying number of Departments, these functions would not be an appropriate choice. The developer cannot attach any SCL code to the LISTx window: returned values must be examined and the function possibly be recalled if an inappropriate choice is made. Likewise, display of the LIST entry's items cannot be constrained: if only certain departments were appropriate, given some other screen field's value, LISTC could not restrict its display to only the appropriate items (using, perhaps, a WHERE statement).

Another drawback is that the list is effectively static. Changes to the list entry must be made manually. Finally, the separation of program and list catalog entries may be viewed by some people as a drawback, especially if the entries are in separate catalogs. The functions discussed below address some of these drawbacks.

Method 4: DATALISTx Functions

The DATALISTC and DATALIST functions offer the most flexibility of the tools reviewed so far. Rather than LISTx's reference an external LIST, MENU, or HELP catalog entry, the DATALISTx functions build their selection lists from SAS datasets. This means that reference datasets can be dynamically updated and the AF

program will automatically display the changes. The functions can use datasets in native SAS format or in a foreign (ORACLE, RDB, etc.) format via SAS/ACCESS. WHERE restrictions can be placed on the dataset, unlike the LISTx functions described earlier.

The DATALIST windows behave the same as SHOWLIST and LISTx. The user scrolls through the list and presses Enter when the cursor rests on the desired selection. A mouse click achieves the same end. The developer can specify exit from the window as soon as a selection is made (the default), or the user can be forced to specify an End or Cancel command to exit.

Exhibit 5 illustrates an application using the DATALISTC function. The calling program's DISPLAY screen has "filter" and Department fields. The filter contains a logical condition constraining the choices which will be displayed by DATALISTC. Remember this paper's display constraints - in a real application the display would be more descriptive and the program would probably validate FILTER before using it.

The Exhibit shows FILTER already filled in: the DATALISTC function will only display observations whose SIZE value is over 100.

DISPLAY (user has specified a filter)

Enter filter: size > 100 _____
 ^
 Field FILTER

Department: ____ ← Field DEPT

SOURCE

```
init: dsid = open('ref.dept','i');
      return;

main: if filter ^= ' ' then
      rc = where(dsid, filter);
      dept = datalistc(dsid,
        'dept desc size loc');
```

DATALISTC Window

DEPT	DESC	SIZE	LOC
088	Personnel	340	SanDiego
091	Personnel	120	Raleigh
100	Claims	135	Richmond

Exhibit 5 Using the DATALISTC function

Review: Advantages The benefits of a dynamic link to the SAS dataset have already been noted. In addition, the DATALIST functions have another, significant advantage over other techniques reviewed so far. The second parameter may specify *several* variables to display in the pop-up window (the first variable is the one whose value will be returned by the function). Earlier methods were limited to a single display field. The extra fields (DESC, SIZE, and LOC) provide useful annotation. Rather than have a potentially obscure 3-character code, the user sees a description of the department as well as other fields which may influence the selection process.

Like the LISTx functions, DATALISTx enables multiple applications to share lists. Another advantage of DATALISTx is that it is relatively easy to code. The only extra effort involves the OPEN and, possibly, WHERE functions. And, of course, the developer must be aware of the variable names and data types of the dataset being displayed.

Review: Drawbacks DATALISTx is subject to many of the negatives noted in earlier sections. Its ability to return only one value limits its usefulness. The user has very little control of the selection process while the window is open. The location, color, and size of the window can be difficult, if not impossible, to modify. The window's presentation may be in conflict with the rest of the application's look and feel.

Method 6: Extended Tables

Let's review the general characteristics of the tools reviewed so far. On the positive side, they are easy to code, offer transparent access to changing data, and can share lists stored as catalog entries or SAS datasets.

They also have several drawbacks. Their "look and feel" may conflict with the rest of the application, and may not be easily modified. Another negative is that with the exception of the multiple-choice hard-coded list (Method 1), they allow only a single selection. This section discusses the use of Extended Tables as selection lists. They are extremely flexible and come only with the cost of extra coding.

Extended tables are PROGRAM entries which allow the developer to repeat a row of SCL fields in a window. The fields may be any combination of user input, SAS datasets, and external files. The appearance of the table is not dissimilar to that of FSVIEW screens: the display may be scrolled, with as many lines from the data source as possible shown in the window. A particularly useful feature of extended tables is their ability to serve as selection lists. This allows the developer to have the benefits of both selection lists *and* being able to attach SCL to the list.

- [1] A request for a selection list is issued from a program: the user may click on a button; a field may be blank or invalid; or the user may want to change a valid value. The program requests execution of the extended table.
- [2] The extended table program begins execution. It does some "handshaking" (parameter passing) with the calling program, opens the lookup dataset, and sets up the extended table.
- [3] The table screen fills up with observations from the lookup dataset, one observation per screen row.
- [4] The user moves through the list, possibly marking a row to select. When the user requests a return to the caller program, the extended table retrieves the desired value (if any) and returns it to the calling program. The new value is now available to the caller for display and/or manipulation.

```
main: if modified(dpt) then
      call display('sel',dpt);
```

```

entry d $3;
init: dsid=open('mast.dlist','i');
    call set(dsid);
    call setrow(0,1,, 'y');
    return;

main: if _status_ = 'E' then do;
    which=selected(1);
    if which > 0 then do;
        rc = fetchobs(dsid,which);
        rc = close(dsid);
        call exccmd('end');
        end;
    end;
    return;

term: return;
putrow: return;
getrow: if fetchobs(dsid, _currow_)
        ^= 0 then call endtable();
return:

```

22

[3] Statements in the GETROW label are executed until the table is full. FETCHOBS reads from DLIST. When end-of-file is reached, the ENDTABLE function completes the table.

Exhibit 8 gives a rough idea of what the screen's appearance when a selection is made. The selected row is redisplayed with the error color and video attributes specified in the ATTR screen. In this example, the row is shown in reverse video.

Exhibit 8 Selection List in Use

The modified calling program is shown in **Exhibit 9**. Extra fields have been added to the screen and to the parameter list in the DISPLAY function. Again, the order and type of these parameters is crucial. They *must* agree in order

Dept. Code: _____ « Flds D1, DTXT1
 _____ « Flds D2, DTXT2
 _____ « Flds D3, DTXT3

```
main: if modified(d1) | modified(d2) |
      modified(d3) then
      call display('sel',
                  dpt1-dpt3,
                  dtxt1-dtxt3);
```

and type with the parameters in SEL's ENTRY statement. In Exhibit 10, the modified SEL catalog entry:

```

entry d1-d3 $3 txt1-txt3 $20;
init: dsid=open('mast.dlist','i');
      call set(dsid);
      call setrow(0,3,,'y');
      array d(3) $;
      array dt(3) $;
      return;

main: if _status_ = 'E' then do;
      nchosen = nselect();
      if nchosen > 0 then
        do i = 1 to nchosen;
          row2get = selected(i);
          rc=fetchobs(dsid,row2get);
          d(i) = dept;
          dt(i) = descrip;
        end;
      rc = close(dsid);
      call exccmdi('end');
      end;
      return;
term: return;
putrow: return;
getrow: if fetchobs(dsid, _currow_)
        ^ = 0 then call endtable();
        return;

```

23

- In INIT, ARRAYs are defined to simplify assignment of values to the program's parameters.
- The SETROW call now specifies a maximum of three choices.
- In MAIN, when a request to terminate is issued NSELECT identifies the number of selections. If selections were made, the SELECTED -> FETCHOBS -> assignment logic of the earlier example is again employed, this time in a DO loop.
- When all NCHOSEN selections have been processed, execution of SEL terminates. The values of the D and DTEXT arrays are passed to the caller and made available for display and/or manipulation.

Review: Advantages The second example clearly demonstrates the advantages of using extended table selection lists. They allow multiple selections, can return more than one value from a selected row, and will always reflect the current contents of the dataset.

Earlier techniques such as LISTx and DATALISTx allowed little control over window placement, color, and menus. Extended tables, however, are completely configurable, and are easily made consistent with the look and feel of the rest of the application.

Possibly the most compelling reason for their use is the ability to have SCL code in the background. This enables the developer to: warn the user to make at least one selection; compute totals of, say, Department size and display a running total as selections are made; and so on. The possibilities are limited only by the demands of the application's users.

Review: Drawbacks Just as the benefits of extended tables are obvious so is their principal drawback. The coding effort for a well-designed list is significant. Bear in mind that the sample code is bare-bones: we did not check for successful completion of the OPEN, nor did we check to see that the dataset had at least one observation. These and other tests are required for sound applications and time-consuming to implement.

The documentation requirements for such programs are also greater than for simpler tasks. While not a drawback *per se*, it is yet another task placed in the hands of the developer.

Summary

This paper has quickly toured the more popular methods of implementing selection lists. No single method is always the most appropriate: considerations such as the rate of change of list elements, importance of consistent look and feel, number of selections, degree of underlying control during selection, and other issues will all influence adoption of a method. The chart in Appendix A summarizes some of these issues.

Contact

Comments and questions are welcome. Contact the author at: Trilogy Consulting Corporation, 102 Westbury Drive, Chapel Hill NC 27516-9154 (919) 361-2243.

Appendix A: Comparison of Methods					
Feature	Hard-Coded	SHOWLIST	LISTx	DATALISTx	Extended Tables
Skill level to implement	Low	Low	Moderate	Moderate	High
Multiple selections?	Yes (if no Choice Group)	No	No	No	Yes
Ease of coding	High	High	High	High	Low
Presentation same as rest of app?	Yes	No	No	No	Yes
Recompile to reflect list updates?	Yes	Yes	No	No	No
List separate from code?	No	No	Yes	Yes	Yes
Maximum number of list items	no limit	13	no limit	no limit	no limit
Default: exit when selection made?	No	Yes	No	No	No
Choice immediately available to SCL?	Yes	No	No	No	Yes
Display multiple vars. in list?	Yes	No	No	Yes	Yes
Return multiple vars. from list?	N.A.	No	No	No	Yes