Data Selection Techniques in SAS/AF®: Making Users' Lives Easier

Scott Rompala
Computer Sciences Corporation
Professional Services Group

ABSTRACT

Recently, the trend in software development seems to be inducing the user to rely less and less upon documentation to determine what values belong in the fields of an application. Instead, when running an application, a user often needs to make choices, either in data entry or selecting parameters for obtaining specific output. On the data entry side, the kind programmer will add functionality to an application so that the user can not make errors which might be costly to fix later. On the output side, the programmer can aid a user in choosing value(s) so that he/she does not have to remember the correct value(s) to cause the results that he/she wishes. Fortunately, SAS/AF presents a programmer with the concept of the selection list, which remedies both of these problems. This paper will demonstrate seven different SAS/AF techniques (List Attributes, List Entries, Choice Groups, SHOWLISTs, DATALISTs, Extended Tables, and SCL Lists) for implementing selection lists in an interactive format, and will attempt to help a programmer, who is creating an application with the user's needs in mind, decide which to use in a specific situation.

INTRODUCTION

Your manager assigns you the task of building a user interface in SAS/AF. The application should allow users to enter data and to perform many administrative and managerial functions. Your manager stresses the fact there will be many users on the system and he does not want to print manuals for all of them. In other words, you must make the application so user-friendly that anyone who uses the application will not need a manual. This implies that the application has to provide a great deal of help for the inexperienced users. Fortunately, you have a host of techniques in SAS/AF that will allow you to create this application according to your supervisor's specifications. This paper will explain only seven of the numerous techniques that you can utilize and compares their effectiveness by reason of their strengths and weaknesses. This will hopefully aid you in making a decision between the various techniques. Since this paper is not a tutorial in SAS® Screen Control Language (SCL), we will not discuss the means of creating several of the techniques and many of the less relevant options.

THE LIST ATTRIBUTE TECHNIQUE

The simplest technique to aid the user in making data choices is the List Attribute. The List Attribute is associated with each data field on a screen and is located within the attribute window of a data field. Use of List Attributes can present the user with a very simple selection list of values.

To present the user with a list of values, place the values in the List Attribute for the screen field. An example would be if you have a screen field that is called CITY. Since your firm only covers certain cities, you wish to provide the user with the ability to choose the name of a city, rather than typing a possibly invalid one. In either the PROGRAM or FRAME data entry field, you could create the List Attribute as:

List: Dallas Austin Houston Lubbock Waco

The user can type a ? in the field to get a selection...
Using the List Attribute as a method of assisting users is very simple. This method requires little programming work. Presenting users with data choices in this manner does not require Screen Control Language (SCL) so you do not have to compile the entry.

Unfortunately, this method has drawbacks. Using only the List Attribute, it is not possible for the user to make more than one selection. Therefore, you could not implement a multi-selection list easily.

Another problem occurs when the user enters a ? to bring up a selection list. He will see the same selection message every time. You are not able to customize the message presented with the selection list using only the List Attribute. Additionally, if the user types in an invalid value, he will always view the message "value is not in list of valid values." You are not able to change this message to inform the user about the nature of his error.

Another disadvantage is that you have a limited amount of space for possible values in the List Attribute. PROGRAM and FRAME entries have less than 50 characters of space. This can be an important problem if you want to give the user a larger universe of choices.

A good example is if a CITY field contains a city in Texas. There are far more cities in Texas than can fit in the small amount of space in either a PROGRAM or a FRAME data entry field, making the simple List Attributes a poor choice with large amounts of selections.

Finally, if you use this technique you cannot easily change the values of the list while the application is running. Hard-coding in this manner leads to more work if you wish to make changes at a later time.

**THE LIST ENTRY TECHNIQUE**

Use of the List Entry technique is more complicated than the simple List Attribute. The List Entry is an AF catalog entry that contains a list of either character or numeric values. The list can be as large as needed. This entry does not function well by itself, but when paired with one of two AF constructs, either the List Attribute or the LISTC or LISTN function, it can become an unlimited length selection list.

When you create the List Entry, you have several useful options for formatting and displaying your data. These include specification of informat and formats, justification, and the fileref from which you can optionally populate the List Entry.

The contents of the List Entry, when paired with the List Attribute, is displayed when the user types a ? in the data entry field. When you create the PROGRAM or FRAME data entry field, you can make the values in the List Entry available to users by placing an equal sign and the name of the List Entry in the List Attribute field in the following manner:

```
List:  = cities
```

You can allow users to select more than one value by placing the maximum number of selections available next to the List Entry.

```
List:  \? 3 = cities.list \n```

When you use the List Entry with the LISTC or LISTN function, you do not have to associate the values with a screen field. You can display them at the touch of a key or a click of a mouse. The LISTC or LISTN
function can also show a custom instructional message to the user. The following example will present the user with many cities in Texas and return the user's choice.

cval=listc('cities.list', 'Please choose one city:');

The List Entry is not a difficult method of displaying a selection list. This process requires very little coding time, so that it is ideal for the novice programmer. The List Entry can be extremely long and is easy to modify. This can add flexibility that values of fixed length would lack. Additionally, while other methods tie selection lists to a specific field, this method, when used in conjunction with the LISTC or LISTN function, allows you to offer the user a selection list with the touch of a key. The user can select several values when using the List Attribute in conjunction with the List Entry. You can also query the user for multiple choices when you use the List Attribute with the List Entry.

However, there are several disadvantages to using List Entries. The most important is that you must hard-code the values in the List Entry, that is, the stored values cannot be changed at run-time. Even if you populate the entry from a dataset, the dataset values are transferred to the List Entry when the entry is built. If you wish to add a city to a selection list, and you had stored the cities in a dataset, you would have to update the dataset and recreate the List Entry. This does not provide much flexibility from within the application. Also, although the List Entry can both provide a custom instructional message to display to the user and return several values from the selection list, these functions are not available concurrently. You can provide multiple selections with the List Attribute, and an instructional message by the LISTC or LISTN function. If you wish the user to be able to select several values, you must forego the selection message, and vice versa.

THE CHOICE GROUP TECHNIQUE

Choice Groups are a method of combining several screen fields (e.g. character fields, push buttons, icons, etc) into stations to become a selection list. The value of the Choice Group is the value of the station that the user has selected. Because the stations can be virtually any length, these are often data choices that take up much of the screen.

For example, you create a data entry screen to let a user specify an activity that was performed. To accomplish this, you decide to use a Choice Group.

Each of the attribute windows for the Choice Group will look similar to the following:

The SCL to determine which station the user chose is simple.
Using a Choice Group can add "meat" to your choice. Instead of a list of brief values, you can create more descriptive ones as well as more graphical values. Choice Groups are particularly useful when you have extra space on the screen and you would like to expand the length of your values.

However, since you place the Choice Group values directly on the screen, you can only utilize a limited number. Setting up hundreds of values is not a viable option when implementing Choice Groups. Since the Choice Group values use permanent space on the screen as opposed to the temporary space of a pop-up window, you have little space for the rest of the application entry. The non-uniform procedure which is so handy for allowing the values to be more flexible makes it more difficult for you to populate a Choice Group with dataset values. Finally, managing the values takes some work, both in augmenting the ones on the screen, and modifying the SCL.

THE SHOWLIST TECHNIQUE

The SHOWLIST function is a SCL function that displays a list of up to twelve values and an instructional message, and returns the user's choice to the application.

In our example application, you have a screen that requires the month on which each building was erected. Since there are only twelve months, the you decide to use the SHOWLIST function.

```scl
select (group);
  when ('CHILI')
    activit2='ate chili';
  when ('HORSE')
    activit2='have a sore bottom';
  when ('HAT')
    activit2='have hat and boots';
  when ('COWBOYS')
    activit2='watched Bills lose';
  when ('DALLAS')
    activit2='saw Southfork Ranch';
  when ('ACCENT')
    activit2='said Y’All';
end;
```

```scl
month=showlist('JANUARY', 'FEBRUARY', 'MARCH', 'APRIL', 'MAY', 'JUNE', 'JULY', 'AUGUST', 'SEPTEMBER', 'OCTOBER', 'NOVEMBER', 'DECEMBER', 'Choose one month');
```

Whatever month the user chooses will be stored in the variable MONTH.

This style of providing selection lists is very valuable because it is a quick and dirty way of presenting a user with several, possibly unrelated, data choices. You can provide the user with twelve values at run-time from SCL with ease. You can invoke the SHOWLIST function without an accompanying screen field. You can even change the instructional message to one that is more relevant than the default.

The SHOWLIST function is a very simple technique to implement, but also very simple in the actions it can perform. The values presented in the SHOWLIST function can not easily be pulled from a dataset, and only one selection is possible. The most significant problem, however, is that you can only present the user with twelve values. This is severely limiting, since often you will wish to offer your users more than twelve options.

THE DATALIST TECHNIQUE

The DATALISTC or DATALISTN function provides you with the ability to fill a selection list directly from a dataset. The function will create a pop-up window which returns values that the user has chosen.
The DATALISTC or DATALISTN function requires a little extra SCL in order to determine the dataset identifier.

```sas
dsid=open('cities');
call wregion(5,10,10,40);
cvals=datalistc(dsid, 'city', 'Please choose up to 3 cities:','Y',3);
rc=close(dsid);
```

The preceding example will display the contents of the CITY variable for all observations of the CITIES dataset. The application encourages the user to choose up to three of the values.

```sas
BUILD: DISPLAY NEW.PROGRAM (E)
Command ===>
  Select Date
  Command ===>
    Please choose up to 3 cities:  
      - Ft Worth
      - Dallas
      - Austin
      Lubbock
```

This method of displaying selection lists has many advantages. Since the list is being filled from a dataset, you can change the dataset at any time to reflect new or modified values that the user can select. The DATALISTC or DATALISTN statement is very valuable because it is a one line statement that will present a pop-up window (saving space on the screen that Choice Groups use), display the values in a dataset, and return the user's choices. In order to achieve this level of functionality in most other methods, you would have to use more code.

Furthermore, you can display several variables in the pop-up window, as opposed to techniques such as the List Entry where it is only possible to display one. If the user has more information in the form of additional variable values when making a decision, it stands to reason that he/she will be able to make choices more easily.

Finally, you do not have to associate the DATALIST technique with a particular screen field, freeing you to create a selection list when you need it, instead of waiting for a user to request it.

On the other hand, there are several drawbacks to using the DATALIST technique. Unlike several of the other methods that we have discussed, using DATALISTS requires more programming effort to support the selection list. You must create SCL in order to open, close, sort and subset the dataset from which the values will be taken. It also is not possible to remove the command line and the dialog options from the DATALIST without programming effort.

Sizing the window correctly is not always performed automatically by the SAS System. You must use the WREGION statement (or other means) to create the pop-up window size that will display all of your data correctly.

Finally, you have very little control over the method of selection and the validation of the selection from within the DATALIST until the user selects a value. For example, you wish to create a dataset containing a list of cities. You display it as a selection list. Then, you want the user to choose a city which had been the site of the Olympic Games. If the user chose an incorrect city, the application would not be able to take any action until the value had been returned and the window closed. This is not a very versatile way to validate data.

THE EXTENDED TABLES TECHNIQUE

An Extended Table is a section of a PROGRAM or FRAME entry that contains a set of screen fields. These screen fields are representative of the variables in a dataset which will populate the table. A user can scroll the table, displaying as many observations from the supplying dataset as possible. You can use Extended Tables for various reasons. We will discuss the use of Tables as selection lists here. The process for creating Extended Tables is somewhat involved and will not be addressed.

An Extended Table could be used for a portion of an application in which a user must choose reports that he wishes to execute. In our example application, the screen would show the name of the report, a short description, and the page number to place the report.
The built (but not executed) table would look like the following:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Page Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Description</td>
<td>Page Number</td>
</tr>
</tbody>
</table>

The finished table might have ten observations, with only five displayed at a time.

Choosing reports from the list would cause particular actions to take place, depending on the SCL behind the Extended Table. A selected report could be deleted, added to another list, or run. You could allow the user to make as many selections for which you have provided.

Extended Tables are very practical when presenting several variables from within a dataset. The mask row (the row containing the variable characteristics for the table) can accommodate variables to the length of the screen. It is not uncommon for an Extended Table to display three to ten variables.

However, one of the most significant reasons for using the Extended Tables technique is the level of control that you can possess. The SCL statements and processing behind the Extended Table allow you to control just about every facet of the selection effort. This allows you to determine the validity of the selections, as well as the ability to act in response to a selection.

Since the Extended Table is actually part of the PROGRAM or FRAME entry, it can exist during the entire display of the entry. Choosing values from the list will not close the Table or make it inoperable, unless specified by you. The user can choose one or several values from the list, depending on the specifications of the system. Finally, if you create a dynamic selection list, there is no limit to the number of values that you can display from a dataset. If the number of values is greater than the number that can be displayed on the screen at one time, the user can scroll the Table to view the other values. This dynamic operation allows for maximum flexibility with large amounts of variables.

There are some detriments to using the Extended Tables technique. The most serious is the amount of coding that you must do. While several other techniques require only one line of code, Extended Tables require code to open and close the selection dataset, to link the values to the fields in the Extended Tables, and to perform any operations which should occur when values are selected. It is not unusual to write ten to twenty lines of code just for the basic operations of the Table.

Another disadvantage to using Extended Tables is that they use permanent space on the screen. You can compensate by making the Table smaller, so that fewer rows are displayed. However, placing several other fields on the screen besides the Table could be a design problem as well as making the screen too busy.

Finally, Extended Tables are composed of screen fields. Therefore, it is not possible to create an Extended Table without those fields.

**THE SCL LIST TECHNIQUE**

SCL Lists are a data structure available in SAS versions 6.08 and higher (version 6.07 on some platforms). Basically, they are either permanent or temporary lists of values. Each value in the list can be numeric, character, or another SCL List. Because they are a “runtime” version of the standard LIST entry, you are able to insert values into the SCL List, as well as delete values from it, from within the SCL. This data construct provides a very flexible method for dealing with values. However, as with most data constructs, the greater the flexibility, the more work you must do to handle the code behind it.

Although SCL Lists can consist of different types of values, we will only concentrate on creating a simple SCL List, consisting of either numeric or character
values, and displaying it as a selection list.

There are two ways of populating a SCL List that we will examine: the INSERT method, and the LVARLEVEL method.

Both methods require a list identifier, hereafter noted by the variable LISTID, which references a specific, existing SCL List.

The INSERT method will insert either character or numeric values into a SCL List. The following example illustrates populating an SCL List with the letters 'A,' 'B,' and 'C.'

```sas
do i = 'A','B','C';
    L1STID=insertc(LISTID,i);
end;
```

In a similar manner, you can populate a SCL List with numeric values by using the INSERTN function.

The LVARLEVEL method will insert all the values of a dataset into a SCL List. This method requires, as well as an existing list referenced by LISTID, an existing dataset referenced by DSID.

The following example will populate the SCL List with a list of several cities in Texas.

```sas
nlvl=0;
/*SAS sets this to the number of unique values found*/
rc=lvarlevel(dsid,'city',nlvl,listid)
```

Now that the list is populated, two SCL functions will display the SCL List, and return the user's choice. The POPMENU function displays the list of values and returns the position in the list of the value that the user has chosen. The GETITEMC function returns the value at a certain position in the list.

```sas
position=popmenu(LISTID);

city=getitemc(LISTID,position);
```

The SCL List method is a very powerful way of displaying a selection list. If dynamically created, the list can be as long as necessary. It can be filled directly from a dataset, as the Extended Tables and DATALIST techniques can, but also can be filled from non-dataset values, which they can not do easily. Additionally, when using monitors with good graphic capabilities, the background is much cleaner than the standard one provided by a DATALIST. The SCL List is not joined to a screen field, so it can be called by the touch of a key or mouse.

One of the primary barriers to using the SCL List is that it requires a high degree of skill to master the concepts behind the SCL List as compared to the other methods discussed here. Manipulating list identifiers and items within the list can be confusing to a SAS/AF novice.

Although creation of the list is very flexible, the POPMENU function does not allow very much flexibility for the user. You are not able to display an instructional message with the SCL List and the user can only make one selection. A multiple selection list is only available if the SCL code is called multiple times.

**COMPARISON**

You should consider the following "rules" when creating a selection list in an application.

- Use the simplest technique for your situation.

It should not be necessary to use the SCL List technique if you are displaying 10 values. The time
spent in implementing the SCL List could be wasted.

- Use the technique that requires the least amount of CPU and I/O time.

If your selection list contains the values from one variable, is it worth the effort to implement an Extended Table on a screen by itself, when a DATALIST would accomplish the same result? The CPU time for processing the Extended Table and I/O incurred by reading from a dataset might be wasted.

- Keep your list of values that will populate the selection list as flexible as possible.

It would be inefficient to hard code a list of cities into a List Attribute when the list could very easily grow. Driving the selection list from the data in a dataset could be a better idea.

- On the other hand, consider if it is necessary to data-drive static values.

If a selection list contains the days of the week, months of the year, or other data that has a low probability changing, you might not want to place these values in a dataset. A low-level technique could be easier to implement.

CONCLUSION

We have defined seven techniques for creating selection lists. Some are very simple methods, and some are very complex. The next time you create a selection list, remember the "rules" listed here and weigh the advantages and disadvantages inherent in each of the techniques in order to make your selection list as powerful and efficient as possible, while still maximizing the amount of information that you are providing your user.

ACKNOWLEDGMENTS

The author wishes to thank the staff of the Computer Sciences Corporation Professional Services Group, Inc for their invaluable help in producing this paper.

SAS is a registered trademark or trademarks of SAS Institute, Inc. in the USA and other countries. © indicates USA registration.

The author can be contacted at:

Computer Sciences Corporation
Professional Services Group, Inc
Information Systems Division
1301 Piccard Drive
Rockville, MD 20850

Voice: (301) 258-5300
FAX: (301) 258-6878
## Appendix

### Comparison Table of 7 Data Selection Techniques

<table>
<thead>
<tr>
<th>Function</th>
<th>List Attributes</th>
<th>List Entry</th>
<th>Choice Groups</th>
<th>SHOWLIST</th>
<th>DATALIST</th>
<th>Extended Tables</th>
<th>SCL Lists</th>
</tr>
</thead>
<tbody>
<tr>
<td>Easily filled from dataset?</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Built-in selection message?</td>
<td>No (simplest case)</td>
<td>No (Yes)*</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Number of variables displayed?</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Multiple</td>
<td>Multiple</td>
<td>1</td>
</tr>
<tr>
<td>Number of selections allowed?</td>
<td>1</td>
<td>Multiple (1)*</td>
<td>Multiple</td>
<td>1</td>
<td>Multiple</td>
<td>Multiple</td>
<td>1</td>
</tr>
<tr>
<td>SCL necessary?</td>
<td>No</td>
<td>No (Yes)*</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Expertise level (1-5)</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>Limit on number displayed?</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes, 12</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Runtime/Buildtime population of list?</td>
<td>Buildtime</td>
<td>Buildtime</td>
<td>Buildtime</td>
<td>Buildtime</td>
<td>Runtime</td>
<td>Runtime</td>
<td>Runtime</td>
</tr>
<tr>
<td>Must be associated with screen field?</td>
<td>Yes</td>
<td>Yes (No)*</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Does list use permanent space on screen?</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

*Without parentheses is in combination with the List Attribute construct. Those in parentheses are with the LISTC or LISTN function.*