INTRODUCTION TO SCL LISTS

Roger Staum
SAS Institute Inc.

ABSTRACT

The SCL list is a flexible and powerful data structure which first appeared in version 6.07 of the SAS® system. This paper will help you:

- understand what SCL lists are, and how they differ from other data structures
- decide whether a particular application is a good candidate for an SCL list
- learn SCL list terminology
- classify SCL lists and items by access, type, and attributes
- manipulate SCL lists using special functions and routines
- plan for future enhancements to SCL list technology

INTRODUCTION

An SCL list is an ordered collection of data in the Screen Control Language (SCL) environment. The elements of an SCL list are called items.

SCL lists are not the same as selection lists. A selection list is a window object which prompts a user to select one or more items from a displayed list. An SCL list, by contrast, remains in memory and stores data. The contents of an SCL list may be used to populate a selection list, and items selected from a selection list can be used to fill an SCL list. The functions and routines used to manage SCL lists are distinct from those used to generate selection lists. Throughout this paper, the unmodified term "list" means an SCL list.

Lists have the following features which distinguish them from arrays:

- mixed types - A single list may contain items of various types, including sublists.
- dynamic sizing - The number of items in a list can change at run time.
- easy manipulation - Powerful functions and routines exist which facilitate operations on lists.
- permanent storage - Lists may be stored between sessions and reused.
- indexed and named access - List items may be accessed either by position or by name.
- data sharing - A list can be readily accessed throughout an application or a session by passing a single id number between entries.[1]

The primary source of information for lists is Reference 1, chapters 8 and 17-20. You can find additional information in Reference 2, chapter 4, and obtain on-line assistance by issuing the command **HELP SCL**.

This paper occasionally uses different terminology than do the references, when I believe this makes the concepts easier to understand. I have included the "official" terminology in parentheses.

APPLICATION EXAMPLES

You should consider using lists instead of other SAS data structures when the features listed above are relevant and important to your application. For instance:

- You need to read or write data frequently. (Active lists are stored in memory, so I/O is rapid.)
- Your data are inherently non-rectangular. (Sublist structures are ideal for representing hierarchies, networks, etc.)
- You need to store data across sessions.
- You prefer to reference items using descriptive names, so that you do not need to keep track of positions.
- You want to store data of mixed types in one structure.
- You need to reorder your data frequently.
- You need to pass a lot of information to an application or between entries.
- You need to build a selection list dynamically from an external file.

Here are several examples of applications which might use lists:[2]

- Since program source code is inherently repetitive and hierarchical, you can design a code generator by storing components (steps, statements, keywords, options, etc.) in lists.

- You can use lists instead of the global symbol table to minimize parameter passing between entries, etc. The list-based I/O functions execute faster than the corresponding macro-based I/O functions. The main problem is that the DATA step cannot communicate directly with lists, as it can with symbol tables.
- In an FSEDIT application, you can store a "history" list of observations or WHERE clauses already requested by the user. If the user wants to return to a previously displayed observation or WHERE clause, you can pop a selection list to make it easy. If the nature of the application is such that the user is likely to want to see the same subset of observations across sessions, the list can be stored and reactivated.
- Pmenus are inherently repetitive and hierarchical. The structure of a system of pmenus can be stored efficiently in a list. Individual pmenu elements are implemented as sublists, so you can modify an element in one place, instead of wherever it is used in PROC PMENU code. You can then write an SCL program to read the list and generate PROC PMENU code.
- Several SAS software products and internal applications have been designed using lists, including FRAME entries, the SQL Query window in SAS/ASSIST, and the Technical Support tracking system (NEWTS).

ACCESS

List access

Lists can be either active or stored.

Active lists reside in memory, and may be used in an SCL program. When an active list is first created at run time, the SAS system assigns it a unique numeric identifier, called a listid.

Stored lists may reside in either catalog entries or external files. Lists are stored using the SAVELIST function, and activated using the FILLIST function.

Item access

You can access items by either index (position) or name.

Indexed access identifies an item by its absolute position within a list. The index may be either positive (counting from the beginning of the list) or negative (counting from the end of the list). For example:

Pos. index	1	2	3	4
Neg. index	-4	-3	-2	-1
Value	**Menu**	**Exit**	**S1**	**S2**

Named access identifies an item by an optional name. By default, these names:

- can be up to 200 characters long
- do not have to be unique
- are automatically upcased
- contain no trailing blanks

For example:

Pos. index	1	2	3	4
Neg. index	-4	-3	-2	-1
Name	OBJECT	TAG	CONTAINS	CONTAINS
Value	**Menu**	**Exit**	**S1**	**S2**

In the remainder of this paper, the index will be omitted when a list is displayed. You should assume that the index of the leftmost item is 1, of the item to its right is 2, etc.

TYPES

List types

Active lists can be grouped into 4 types, based on scope and owner.

The scope[3] of a list is either local or global. Local lists are available within the application[4] in which they are created, and are automatically deleted when that application ends. Global lists are available within the entire SAS session in which they are created, and are automatically deleted when that session ends.

The owner of a list is either the system or the user. The SAS system automatically creates and deletes one global list per session and one local list per application. System lists are also known as environment lists. At any point in an SCL program, the user may create and delete local and global lists.

Lists are automatically assigned numeric listids when they are created. Your SCL program should store and reference these listids in SCL variables. Listids for user lists must be passed between entries using parameters, macro variables, or system lists. Listids for system lists can be determined by calling the ENVLIST function.

You may select any one active list, regardless of type, to serve as the current list. The purpose of this special list is to store values:

- chosen by the user from selection lists[5]
- generated by the LVARLEVEL function

Applications start with no list designated as current. You assign and change the current list using the CURLIST function. You may even have no current list, by assigning 0 as the current listid.

Item types

Items may be one of 3 types, depending on the values they store:

- character
- numeric
- list

Character and numeric items are similar to SCL variables. You do not declare character and numeric item lengths, however, since they change in size dynamically. Character values exceeding 200 bytes are truncated without warning.

List items, also known as sublists, are merely pointers to other lists. Since sublists are identified by their listid, these lists must already exist before they can become sublists. The sublist feature is very powerful, since:

- a list may be a sublist of multiple lists.
- sublists may be nested without limit.
- sublists may be recursive. For example, if A and B are two lists, B may be a sublist of A at the same time that A is a sublist of B. A may even be a sublist of itself. Needless to say, you must be very careful with recursive structures!

A special sublist of the local system list is set aside to hold the values of AF or AFA options. This sublist, named _CMDLIST_,[6] contains named items corresponding to:

- the location of the application entry point (LIBNAME, CATALOG, NAME, and TYPE)
- system command options (AWS, PMENU, etc.), if any
- application-specific options defined by you, if any

ATTRIBUTES

Both lists and items have attributes, which determine the operations which may legally be performed on them. A function call which violates these attributes will result in either a non-zero return code or termination of entry execution.[7]

These attributes may be queried (GETLATTR, HASATTR) and modified (SETLATTR) at any time, and may be stored and restored using SAVELIST and FILLIST.

The following tables show these attributes. Arrows indicate default values, which are generally permissive rather than restrictive.

List attributes

Attribute	Description
DEFAULT	All defaults
DELETE ← NODELETE	Can the list be deleted?
UPDATE ← NOUPDATE	Can the list be updated?
FIXEDLENGTH NOFIXEDLENGTH ←	Can the number of items in the list change?
COPY ← NOCOPY	Can sublists be copied?
ANYNAMES ← SASNAMES	Must all items have valid SAS names?
DUPNAMES ← NODUPNAMES	Must all item names be unique?
FIXEDTYPE NOFIXEDTYPE ←	Can items within the list change type?
CHARONLY NOCHARONLY ←	Must all items be character?
NUMONLY NONUMONLY ←	Must all items be numeric?

Item attributes

Attribute	Description
DEFAULT	All defaults
DELETE ← NODELETE	Can the item be deleted?
UPDATE ← NOUPDATE	Can the item be updated?
FIXEDTYPE NOFIXEDTYPE ←	Can the item change type?
ACTIVE ← INACTIVE	Is the item available for selection in a POPMENU?
WRITE ← NOWRITE	Should the item be included when the list is stored?

SCL LANGUAGE ELEMENTS

General information

Most of the SCL language elements which manage lists are functions. I have grouped these elements into categories to help you learn them. When an element seemed to fit multiple categories, I chose the category based on how I thought it would be used most frequently.

Several functions come in families of three. They end with C, L, or N, depending on whether they refer to character, list, or numeric items. You may find that you need to use the ITEMTYPE function first to determine the correct family member to use.

Since References 1 and 2 cover syntax comprehensively, I have chosen to focus, instead, on:

- concepts
- similarities and differences
- subtle details
- examples

The examples in this section are based on an application which stores pmenu structures.

Creating lists

MAKELIST creates a user list, returning a listid. You can specify the scope of the list (default: local) and the initial number of items (default: 0). Any initial items are numeric, missing value, and unnamed. For example:

```
c1 = MAKELIST();
/* local, 0 items */
a1 = MAKELIST(4,'G');
/* global, 4 numeric items */
```

MAKENLIST is similar to MAKELIST, except that you provide names for all initial items, of which there must be at least one. For example:

```
c1 = MAKENLIST('L','OBJECT','TAG');
/* local, 2 named numeric items with missing
values */
```

COPYLIST copies the contents of a list into a new or existing list, returning a listid.

```
c2 = COPYLIST(c1);
/* c2 is an exact copy of c1, with respect to both
attributes and contents. The only difference is
the listid. */
```

List attributes, types, and length

HASATTR reports whether a list has a particular attribute, and returns a value of 0 (false) or 1 (true). For example:

```
rc = HASATTR(c1,'DELETE');
/* The value of rc will be 1 if the attribute has
the default value. */
```

GETLATTR returns a character string containing all the attributes of a list. For example:

```
attrs = GETLATTR(c1);
/* The value of attrs will be the string
   DELETE UPDATE NOFIXEDTYPE
   NOFIXEDLENGTH ANYNAMES
   DUPNAMES NOCHARONLY
   NONUMONLY COPY
if the attributes have their default values. */
```

SETLATTR changes the attributes of a list, and returns a code indicating success or failure. For example:

```
rc = SETLATTR(c1,'FIXEDLENGTH
CHARONLY');
/* Changes the attributes of c1 so that all items
must be character, and items may not be added
or deleted. */
```

CURLIST returns the listid of the current list, and allows you to designate a different list as current. For example:

```
current = CURLIST();
/* Identifies the current list. */
current = CURLIST(c1);
/* Identifies the list which used to be current,
and makes c1 the current list. */
```

The current list reverts to 0 (null listid) when the application ends.

ENVLIST identifies a system (environment) list, returning a listid. For example:

```
global = ENVLIST('G');
```

LISTLEN reports the number of items in a list, returning an integer. It does not count the items in sublists. For example:

listid: c1

Name	OBJECT	TAG
Type	C	C
Value	Command	TOP

```
len = LISTLEN(c1);
/* The value returned to len is 2. */
```

Creating items

COPYLIST can be used to append or merge items from a source list to a target list. For example:

listid: **c1**

Name	OBJECT	TAG
Type	C	C
Value	Command	TOP

listid: **m1**

Name	OBJECT	TAG	CONTAINS	CONTAINS
Type	C	C	L	L
Value	MENU	EXIT	S1	S2

append = COPYLIST(c1,'N',m1);

listid: **append**

Name	OBJECT	TAG	CONTAINS	CONTAINS	OBJECT	TAG
Type	C	C	L	L	C	C
Value	MENU	EXIT	S1	S2	Command	TOP

merge = COPYLIST(list3,'N MERGE',list1);

listid: **merge**

Name	OBJECT	TAG	CONTAINS	CONTAINS
Type	C	C	L	L
Value	Command	TOP	S1	S2

The INSERT family of functions creates a new item by specifying an insertion position, returning a listid. Insertion pushes items to the right (higher index) to make room. For example, if c1 exists but is empty:

c1 = INSERTC(c1,'TOP',1,'TAG');
c1 = INSERTC(c1,'Command',1,'OBJECT');

listid: **c1**

Name	OBJECT	TAG
Type	C	C
Value	Command	TOP

LVARLEVEL fills a list with the unique, formatted values of a variable from a SAS data set, returning a code indicating success or failure. If the listid is not specified, it defaults to the current list. The 3rd argument must be a numeric variable, initialized to 0, in which LVARLEVEL will place the number of values found.

For example, suppose you have opened the following dataset:

dsid: **flights**

DATE	DEST	BOARDED
01MAR90	LON	198
01MAR90	FRA	207
01MAR90	LON	205
01MAR90	PAR	138
02MAR90	LON	147
02MAR90	FRA	176
02MAR90	LON	201
02MAR90	PAR	172

Assume that the values shown above for DEST are formatted, and you execute the following code:

cities = CURLIST();
nvals = 0;
rc = LVARLEVEL(flights,'DEST',nvals);

listid: **cities**

NAME			
TYPE	C	C	C
VALUE	Frankfurt	London	Paris

and the value of **nvals** is 3.

Item attributes and types

HASATTR reports whether an item has a particular attribute, returning a Boolean value. For example:

rc = HASATTR(c1,'DELETE',2);

GETLATTR returns a character string containing all the attributes of an item. For example:

attrs = GETLATTR(s1,3);

If item 3 contains default attributes, then the value of **attrs** is the string::

ACTIVE WRITE NOAUTO NOEDIT DELETE UPDATE NOFIXEDTYPE

SETLATTR changes the attributes of an item, returning a code indicating success or failure. For example:

rc = SETLATTR(s1,' INACTIVE',4);

changes the attributes of item 4 so that a user may not select it if it is displayed by POPMENU.

ITEMTYPE determines the type of an item, returning a 1-character value: C, L, or N. For example:

listid: **m1**

Name	OBJECT	TAG	CONTAINS	CONTAINS
Type	C	C	L	L
Value	MENU	EXIT	S1	S2

```
type = ITEMTYPE(m1,3);
/* returns the value L to type */
```

Changing item order

REVLIST reverses the order of items in a list, and returns a listid. For example:

listid: **m1**

Name	OBJECT	TAG	CONTAINS	CONTAINS
Type	C	C	L	L
Value	MENU	EXIT	S1	S2

```
m1 = REVLIST(m1);  /* ⇆ */
```

listid: **m1**

Name	CONTAINS	CONTAINS	TAG	OBJECT
Type	L	L	C	C
Value	S2	S1	EXIT	MENU

ROTLIST rotates the items in a list, and returns a listid. You can control the direction and magnitude of rotation. For example:

listid: **m1**

Name	OBJECT	TAG	CONTAINS	CONTAINS
Type	C	C	L	L
Value	MENU	EXIT	S1	S2

```
m1 = ROTLIST(m1);  /* ⇄ */
```

listid: **m1**

Name	CONTAINS	OBJECT	TAG	CONTAINS
Type	L	C	C	L
Value	S2	MENU	EXIT	S1

To restore the original order:

```
m1 = ROTLIST(m1,-1);  /* ⇆ */
```

SORTLIST sorts the items in a list, and returns a listid. You can sort the entire list or only part of the list, by value or name, ascending or descending, case-sensitive or case-insensitive, keeping or deleting duplicate items. For example:

listid: **m1**

Name	OBJECT	TAG	CONTAINS	CONTAINS
Type	C	C	L	L
Value	MENU	EXIT	S1	S2

```
m1 = SORTLIST(m1,'NAME');
```

listid: **m1**

Name	CONTAINS	CONTAINS	OBJECT	TAG
Type	L	L	C	C
Value	S1	S2	MENU	EXIT

Changing item values

DESCRIBE fills a list with descriptive information about a SAS file, returning a code indicating success or failure. The list must already contain items with specific names corresponding to file attributes. The SAS file does not have to be opened first. For example:

```
dsdesc =
MAKENLIST('L','ENGINE','LABEL');
rc = DESCRIBE('WORK.FLIGHTS',dsdesc);
```

listid: **dsdesc**

Name	'ENGINE'	'LABEL'
Type	C	C
Value	V608	'International Airlines'

The SETITEM family of functions assigns a value to a new or existing item accessed by position, returning a listid. For example:

listid: **c1**:

Name	OBJECT	TAG
Type	C	C
Value	Command	TOP

```
c1 = SETITEMC(c1,'BOTTOM',2);
```

listid: **c1**

Name	OBJECT	TAG
Type	C	C
Value	Command	BOTTOM

The SETNITEM family of functions assigns a value to a new or existing item accessed by name, returning a listid. For example:

listid: **c1**:

Name	OBJECT	TAG
Type	C	C
Value	Command	TOP

```
m1 = SETNITEMC(m1,'BOTTOM' ,'TAG');
```

Name	OBJECT	TAG
Type	C	C
Value	Command	BOTTOM

Searching lists

Note: None of the following functions searches sublists.

NAMEDITEM searches for an item with a specified name, returning the position of the first instance found. For example:

listid: **m1**

Name	OBJECT	TAG	CONTAINS	CONTAINS
Type	C	C	L	L
Value	MENU	EXIT	S1	S2

```
pos = NAMEDITEM(m1,'CONTAINS');
/* assigns the value 3 to pos */
```

NAMEITEM searches for an item at a specified position, returning, and optionally changing, its name. For example:

listid: **c1**

Name	OBJECT	TAG
Type	C	C
Value	Command	TOP

```
name = NAMEITEM(c1,2);
/* assigns the value 'TAG' to name */
name = NAMEITEM(c1,2,'EXTERNAL
NAME');
/* assigns the value 'TAG' to name, and changes
the name of the item: */
```

listid: **c1**

Name	OBJECT	EXTERNAL NAME
Type	C	C
Value	Command	TOP

The GETITEM family of functions searches for an item at a specified position, returning its value. For example:

listid: **m1**

Name	OBJECT	TAG	CONTAINS	CONTAINS
Type	C	C	L	L
Value	MENU	EXIT	S1	S2

```
sublist = GETITEML(m1,3);
/* assigns the value of s1 to sublist */
```

The GETNITEM family of functions searches for an item with a specified name, returning the value of the first instance found. For example:

listid: **m1**

Name	OBJECT	TAG	CONTAINS	CONTAINS
Type	C	C	L	L
Value	MENU	EXIT	S1	S2

```
sublist = GETNITEML(m1,' CONTAINS');
/* assigns the value of s1 to sublist */
```

The SEARCH family searches for an item with a specified value, returning the position of the first instance found. For example:

listid: **m1**

Name	OBJECT	TAG	CONTAINS	CONTAINS
Type	C	C	L	L
Value	MENU	EXIT	S1	S2

```
pos = SEARCHL(m1,s1);
/* assigns the value 3 to pos */
```

Search function summary table

Function returns → / You specify ↓	Position	Name	Value
Position		NAMEITEM	GETITEM
Name	NAMEDITEM		GETNITEM
Value	SEARCH		

Note that, although there is no single function which returns a name from a specified value, you can perform this task by passing the result of SEARCH to NAMEITEM.

Displaying list contents

POPMENU displays a selection list containing the values of all items in a list, returning the position of the selected item. The selection list includes scroll bars if its length exceeds the screen size. For example:

listid: **s1**

Name	OBJECT	TAG	TEXT
Type	C	C	C
Value	SELECTION	OK	END

```
pos = POPMENU(s1);
choice = GETITEMC(s1,pos);
```

displays a selection list of 3 items:

Selection
OK
END

and captures the user's selection in **choice**.

Note the following POPMENU restrictions:

- allows only one selection from the list
- will fail if the list contains numeric or sublist items
- has no option to display names
- displays all, not just unique, values
- does not return a value directly

The PUTLIST routine displays names, values, positions, and listids for a specified list and its sublists, in the MESSAGE window. It can be very helpful for debugging list program logic. The indentation option, although requiring more window space, makes the output easier to read. For example:

listid **m1**:

Name	OBJECT	TAG	CONTAINS	CONTAINS
Type	C	C	L	L
Value	MENU	EXIT	S1	S2

CALL PUTLIST(m1,'MENU 1: ',1);

MESSAGE

```
List M1: (OBJECT='Menu'
      TAG='Exit'
      CONTAINS=(
            OBJECT='Selection'
                  TAG='OK'
                  TEXT='END'
                  )[5]
      CONTAINS=(
            OBJECT='Selection'
                  TAG='Go Back'
                  TEXT='CANCEL'
                  )[7]
    )[9]
```

Storing and restoring lists

SAVELIST stores a list in a catalog entry or external file, returning a code indicating success or failure. If you save to an SLIST catalog entry, the structure and contents of the list are preserved, and can be restored using FILLIST. If you save anywhere else:[8]

- only character values are saved
- SAVELIST will fail if there are any numeric or sublist items

For example:

rc = SAVELIST('CATALOG',
'SASUSER.PMENUS.SAMPLE.SLIST',c1);

FILLIST can populate an active list with any of the following:

- information from a SAS catalog entry
- information from an external file
- icon numbers
- catalog names in the search path

returning a code indicating success or failure. For example, if you saved a1 to a catalog entry named SAMPLE.SLIST in a catalog called SASUSER.PMENUS, then you can restore it later:

newa1 = MAKELIST();
rc=FILLIST('CATALOG',
'SASUSER.PMENUS.SAMPLE.SLIST',newa1);

Deleting Lists

DELLIST deletes a list, optionally including its sublists, returning a code indicating success or failure. For example, to delete **m1**, but not its sublists:

listid: **m1**

Name	OBJECT	TAG	CONTAINS	CONTAINS
Type	C	C	L	L
Value	MENU	EXIT	S1	S2

rc = DELLIST(m1);
/* lists **s1** and **s2** still exist */

Deleting Items

CLEARLIST removes all items from a list, returning a code indicating success or failure. For example:

listid: **c1**

Name	OBJECT	TAG
Type	C	C
Value	Command	TOP

rc = CLEARLIST(c1);
/* leaves **c1** with the same listid, type, and attributes, but with no items. */

listid: **c1**

Name
Type
Value

DELITEM deletes an item by position, returning a listid. For example:

listid: **m1**

Name	OBJECT	TAG	CONTAINS	CONTAINS
Type	C	C	L	L
Value	MENU	EXIT	S1	S2

m1 = DELITEM(m1,3);

listid: **m1**

Name	OBJECT	TAG	CONTAINS
Type	C	C	L
Value	MENU	EXIT	S2

DELNITEM deletes an item by name, returning a listid. For example:

listid: **m1**

Name	OBJECT	TAG	CONTAINS	CONTAINS
Type	C	C	L	L
Value	MENU	EXIT	S1	S2

```
m1 = DELNITEM(m1,'CONTAINS');
```

listid: **m1**

Name	OBJECT	TAG	CONTAINS
Type	C	C	L
Value	MENU	EXIT	S2

The POP family of functions removes an item from a list by position, returning the value of the deleted item. Items at higher positions automatically drop down to the next lower position. For example:

listid: **m1**

Name	OBJECT	TAG	CONTAINS	CONTAINS
Type	C	C	L	L
Value	MENU	EXIT	S1	S2

```
value = POPL(m1,3);
/*places the value of s1 in value and transforms
m1*/
```

listid: **m1**

Name	OBJECT	TAG	CONTAINS
Type	C	C	L
Value	MENU	EXIT	S2

BACKGROUND EXECUTION

Suppose you have an application for which you would like to use lists, but you need to run that application in a batch or background mode. If the application does not require interaction with a user, then you can implement it in entries of type SCL, which have no display window.

But how do you invoke the application with no command line from which to issue an AF command? There are two possible solutions:

- There is a BATCH option on PROC DISPLAY.[9] For example:

```
PROC DISPLAY C= library.catalog.entryname.SCL
              BATCH;
RUN;
```

- There is a v. 6.09 system option called DMSBATCH, which allows you to include Display Manager statements in your program, as long as they have no interactive requirements.[10]

  For example:

```
OPTIONS DMSBATCH TERMINAL BATCH;[11]
DM 'AF C= library.catalog.entryname.SCL <args>';
```

FUTURE ENHANCEMENTS

Version 6.11

- modest performance improvements
- FILLIST and SAVELIST options which allow intelligent processing of carriage control characters, trimming of trailing blanks, and appending to external files.

Version 7

- significant performance improvements
- simpler syntax, e.g.
  - **m1**[3] ⇔ 3rd item in **m1**
  - **m1**.tag ⇔ item named tag in **m1**

SUMMARY

SCL lists open new horizons in data storage and manipulation. Many tasks can be accomplished with less effort and greater flexibility than ever before in the SAS system. I hope that this paper will help you learn and apply list technology quickly and painlessly.

REFERENCES

1. *SAS Technical Report P-216 - SAS/AF Software, SAS/FSP Software, and SAS Screen Control Language: Changes and Enhancements, Release 6.07*
2. *SAS Technical Report P-242 - SAS Software: Changes and Enhancements, Release 6.08*
3. *SAS Technical Report P-252 - SAS Software: Changes and Enhancements, Release 6.09*

To contact the author, write to:

Roger Staum
SAS Institute Inc.
520 MADison Avenue
New York, NY 10022

---

[1] Multiple sessions or users cannot access the same active list, although they may each read a stored list.

[2] I would like to thank David Biesack, Henrietta Cummings, Frank Eckhausen, David Fahey, Martha Hall, and Jugdish Mistry (SAS Institute), as well as Jeremy Staum (The University of Chicago) and Paulette Staum (American Cyanamid Company), for these examples.

[3] Called "environment" or "class" in the documentation.

[4] For this purpose, an application is defined as

- all entries which execute as a result of a single AF or AFA command or PROC DISPLAY submission
- any FSEDIT/FSVIEW program not nested within another application

[5] CATLIST, COLORLIST, COPYLIST, DATALISTC, DATALISTN, DEVLIST, DIRLIST, FILELIST, LIBLIST, LISTC, LISTN, LVARLEVEL, and VARLIST. The user's selections automatically fill the current list as well as the SCL return variable. When the selections exceed 200 characters, this is especially helpful, since the string will be truncated in the return variable.

[6] See Reference 1, pp. 9-11.

[7] I.e., the infamous message: **WARNING: Program halted.**

[8] The only entry types other than SLIST which can store list contents are those which accept text, such as SOURCE.

[9] See Reference 2, pp. 50-51.

[10] See Reference 3, pp. 2-3. This method also allows you to pass arguments to the application.

[11] Because of a bug, the TERMINAL and BATCH options must also be set. This bug is scheduled to be fixed in a later release.