Abstract
Explanations and examples provide the basis of this tutorial that explains the new screen control language available in Version 6 SAS/AF and SAS/FSP software. The screen control language provides the ability to develop interactive applications. The functions available in the SCL provide the basis for developing conversational systems for end users. Examples will be used to demonstrate how the language works. Differences in SCL usage in SAS/FSP applications and SAS/AF applications will also be demonstrated through examples. Similarities and differences between the SAS Data Step and the screen control language will also be emphasized.

Contents
1. Requirement analysis
2. Design
3. Comparison vs Data Step Language
4. Programming Constructs and features
5. Programming Concepts
6. SAS Data Set Interface
7. Examples
8. Future

Requirements Analysis
For SAS/FSP software, one item of more functionality requested on the SASWARE ballot is cross-field validation in PROC FSEDIT. We will show you how this is done with version 6 and provide examples. In addition to PROC FSEDIT, we will achieve the same functionality with version 6 SAS/AF, now integrated with the screen control language (SCL), eliminating the restrictions of the Version 5 symbolic language (### and ####). We will also cover restrictions of the data step and how much of this is circumvented in version 6.

In addition to functionality version 5 lacked, version 6 also had a few of its own problems: namely running on a much smaller and much slower machine. There must be good keystroke response during execution. The Execution State (running compiled code, more on this later) must be portable to all machines (not even an issue in version 5)

Design
The design algorithm used to achieve the speed and size necessary to run well as well as incorporation of much more extended and extendable functionality: First, PROC FSEDIT and AF work like the programmable data step, where each time the user changes the screen, this is a new transaction record to be processed, and control is given to the user to process the data. Second, the manner in which the user processes the data is through the programming language SCL. Third, the user's SCL statements are handled in two different phases: 1. The compile phase during PROC BUILD or PROC FSEDIT screen modification, 2. the execution phase during AF (which was PROC DISPLAY in version 5) and data set editing in version 6. This differentiation of compile-time and execution-time enables AF and PROC FSEDIT to move much of the processing to set-up or compile-time so execution just performs tasks requested with as little overhead as possible. Making execution time come at any number of arbitrary points later entails saving and porting compiled code which is done in version 6.

Comparison vs Data Step
In comparison to the data step, a transaction record is the screen after the user strikes an attention key, whereas in the data step a transaction record is the next line of input from a file or the next observation from a set for another data set. Variables are implicitly retained in SCL, where they are not retained unless explicitly requested in the data step. More timing control is given to the SCL programmer through reserved labels, which are started at predefined times.

Labels that control the SCL monitor timing, and their parallel in the Data Step:

<table>
<thead>
<tr>
<th>SCL</th>
<th>Data Step</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSEINIT</td>
<td>Data Step started</td>
</tr>
<tr>
<td>INIT</td>
<td>Any observation started</td>
</tr>
<tr>
<td>MAIN</td>
<td>Current observation modified</td>
</tr>
<tr>
<td>TERM</td>
<td>Finish the current observation</td>
</tr>
<tr>
<td>FSETERM</td>
<td>Data step finished</td>
</tr>
</tbody>
</table>

The syntax is the same as the data step, using constructs and features like declaration statements, assignment statements, conditional statements, and do-end blocks.

Programming constructs and features
1. Conditional statement
2. Reserved labels
3. SUBMIT
4. Field modification statements/functions
5. Window functions
   a. AF
   b. FSP
   c. Display format information
   d. Selection list window
   e. PREVIEW window
6. Data Set Functions (see Data Set Interface below)
7. File functions
8. Format functions
   a. INPUT
   b. PUT
9. Macro functions
   a. SYMGET
   b. SYMPUT

10. Extended tables

11. Command line

12. LEAVE and CONTINUE from DO block

13. CONTROL statement to alternate monitor timing. By default, control is given to the program whenever a screen data modification occurs that does not cause an error state. However, we, as the screen builder, can get control more frequently; like every time an 'attention' key is pressed by requesting SCL to drive our MAIN: labels more frequently.

14. Run-Time binding. With the delayed execution or saving of compiled code, there is no compile-time binding to data sets, and many of the functions have run-time variable arguments; such as put, input, and open.

15. Arrays
   a. Operators (=, IN)
   b. Function that accept of lists (min(), max())
   c. Binding with functions like SET,

Programming Concepts

Program Unit In Version 6, program screens, and even FSEDIT sessions can be treated as a subroutine. Through the CALL DISPLAY and CALL FSEDIT statements in SCL, we can branch to these objects, each of which have their own program code, and wait for them to return after setting up any global information we depended on them for. From AF program screen entry 'one.program' we can CALL DISPLAY('two.program'); and 'one.program' treats this in the same fashion an abs() function takes, just sequential execution of the program.

DS Verification We can use data sets to verify the screens, we can now bind the current screen to any number of data sets in any fashion by doing our own data set manipulation within the program source. The list objects or lists of values that data entered is verified against can now come from a data set, and be more dynamic. the data set search can be a linear or binary search depending on how the LOCATE function is invoked.

Maintainability Now the screen can much more easily be broken up into smaller simpler units with fewer 'duties' each. This breakup means the screens are easier for the end-user to understand, and should be much easier for the builder to maintain.

Information Hiding Each screen can display as much information as necessary to the user but only propagate items previous and successive screens need.

Recursion Since each SCL program is an independent reentrant entry object, it can call anything, even another instance of itself via the CALL DISPLAY and Call FSEDIT functions.

Dynamic We can make the programs run based on setup that comes from the screen. In Version 5, INPUT and VARLIST field attributes are an example of checking one field that is based on another. Now we can take this much further by using any logic on any field and having any programmable relationship on any other.

SAS Data Set Interface

OPEN/CLOSE The actual data set that the screen needs does not have to be known at compile time. Simply leave a variable name and whatever that value is will be used as one of the data sets to be used.

Var Info/DS Info We can easily query any information needed about the data set or its variables through the handle returned from open.

Obs Fetch/Update We can fetch, update, append observations through explicit program control.

Var Get/Put When we have an observation, we if we are using the data set for retrieval, we fetch the observation and then query the variables of interest.

SET After we decide on a data set, we can bind the data set variables to the program variables so an observation fetch or update will use the values of the program variables. This is Run-Time binding of a screen to a data set.

LOCATE If we want to do list validation lookup, we can do this on any of the data sets we have opened, against any variable in them, in either linear or binary fashion by specifying parameters to LOCATE.

Examples that follow
1. Value validation
2. Cross field validation

Some programs currently using SCL
1. Version 6.03 menu system
2. Version 6.04 future directions system
3. Full screen confidence interval
4. Full screen project management

Future
1. Interface with Data Step functions
2. Interface between the screens — Parameter passing to replace symget/symput
3. Debugging ability — Source Language Debugger
4. Services window
5. Optional dropping of command and message line
6. A ? to display selection lists
7. Submit from AF to go to appropriate window and return
8. Appointment calendar
9. Calculator
10. Single selection choice groups
11. Switches
12. Multiple selection choice groups
13. New functions for:
   • Cursor positioning
   • Selection Blocks
   • Data Set modification and definition
   • Flat file handling
   • SAS Option query and set
   • User programmable selection lists
   • More default builtin selection lists

Examples
Examples using SCL are on following pages.

Programming Hints, Debugging
   • EMS is highly recommended; without it, AF can go away in order to satisfy memory needs of the procedure.
   • If you need your program screen to continue execution instead of returning for interactive I/O after a submit, use submit continue.

Conclusion
With the structure and power of the language, we can make our screens do nearly anything programmable and still have good response time at the terminal.

Acknowledgements
The authors wish to thank Deva Kumar for his guidance in developing SCL, and Bud Whitmeyer and Brian Schellenberger for their aid in typesetting the paper.

Base SAS®, SAS/AF®, and SAS/FSP® are registered trademarks of SAS Institute Inc., Cary, NC.
Example 1
Value validation from a list

Problem
Given a finite list of of known size and known values, validate the user's entry against the list.

Solution
Put the values into a program array, and search the array every time the user modifies the value.

The program calls the link label indxlkup, which is expected to return the index in the array of the element that equals value from the screen. The link label prtresult sets the value of the message line to indicate whether or not the value is valid based on the index.

Screen
Command == ==
'456' is illegal, Please reenter.

Please Enter Data Value: 456

Program Model
array goodvals[20]
( 1298, 2598, 3465, 4565, 5465,
6787, 7523, 8234, 9734, 10732,
11231, 12534, 13762, 14324, 15634,
16347, 17324, 18492, 19988, 20117 );

array pretty[0:1] $ 25
( 'ok', 'illegal, Please reenter.' );

init:
  return;

main:
  link indxlkup;
  link prtresult;
  return;

term:
  return;

prtresult:
  /*
   * note the trick of using the ordinal value of
   * the 'index = 0' comparison as the array index
   */
  msg = putn(value,'bestS.') || ' ' || pretty[index = 0];
  return;

indxlkup:
  /*
   * sets the variable INDEX
   * to indicate location of matching value,
   * fill in from any of following methods.
   */
  return;

Method 1. Straight linear search of the array. Values do not have to be sorted.

    index = 0;
    do j = 1 to dim(goodvals);
      if goodvals[j] = value then
        do;
          index = j;
        leave;
        end;
    end;
    return;

Method 2. Binary search of the array. Values must be sorted.

    hi = dim(goodvals);
    lo = 1;
    mi = 1;
    index = 0;
    do while(lo <= hi);
      mi = int((lo + hi) / 2);
      if value < goodvals[mi] then
        hi = mi - 1;
      else if value > goodvals[mi] then
        lo = mi + 1;
      else
        index = mi;
        leave;
      end;
    end;
    return;

Method 3. Best Solution Use the IN operator.

    index = value in goodvals;
    return;

Alternate Solution
An alternate solution that requires no programming in the AF or FSEDIT entry is to use SAS Formats and Informats.
The Screen Control Language (SCL)
In Version 6 SAS/AF® and SAS/FSP® Software
Example 2
Value validation from a SAS Data Set

**Problem**
We need to validate user's entry against a list of values that may only be known immediately prior to execution of the screen, and can change between invocations. The values are not of known dimension and value at compile-time.

**Solution**
Maintain the values in a SAS Data Set, which we can update at any time. We apply much of the same technique to validating against the SAS Data Set that we used for an array.

The screen is the same as the previous example.

**Program Model**

```sas
init:
/*
  * get a handle on the data set
  * we are to search against
  * for the comparison values
  */
  dsid = open('valud.ata');
  if dsid = 0 then do;
    _msg_ = 'Unable to open the data set';
    return;
  end;

  vnurn = varnum(dsid,'somevnarn');
  if vnurn = 0 then do;
    _msg_ = 'Variable not in data set';
    return;
  end;

  return;

main:
  link indxlkup;
  link prtrresult;
  return;

term:
/*
  * cleanup, free memory, release the data set
  */
  call close(dsid);
  return;

prtrresult:
  _msg_ = putn(value,'bestS.') 'is' 'pretty'[index = 0];
  return;

indxlkup:
/*
  * sets the variable INDEX
  * to indicate location of matching value,
  * fill in from any of following methods.
  */
  return;
```

**Method 1.** Straight linear search of the data set. Values do not have to be sorted.

```sas
indxlkup:
  index = 0;
  do i = 1 to nobs(dsid);
    if fetchobs(dsid,i) = 0 then
      if getvarn(dsid,vnum) = value then
        do;
          index = i;
          leave;
        end;
      end;
  end;
  return;
```

**Method 2.** Binary search of the data set. Values must be sorted.

```sas
indxlkup:
  hi = nobs(dsid);
  lo = 1;
  mi = 1;
  index = 0;
  do while(lo <= hi);
    mi = int((lo + hi) / 2);
    if fetchobs(dsid,mi) ^= 0 then
      do;
        _msg_ = 'Unable to retrieve observation';
        stop;
      end;
    dsval = getvarn(dsid,vnum);
    if value < dsval then
      hi = mi - 1;
    else if value = dsval then
      do;
        index = mi;
       leave;
      end;
    else
      lo = mi + 1;
    end;
  end;
  return;
```

**Method 3 - Best Solution with unsorted data.** Use the builtin function LOCATE.

```sas
indxlkup:
  index = locaten(dsid,vnum,value);
  return;
```

**Method 4 - Best Solution with sorted data.** Use the builtin function LOCATE.

```sas
indxlkup:
  index = locaten(dsid,vnum,value,'ascending');
  return;
```
The Screen Control Language (SCL) in Version 6 SAS/AF® and SAS/FSP® Software
Example 3
Cross Field Validation

Problem

Cross Field Validation
The laboratory of XYZ hospital need a Lab Report System for their data entry staff. The requirements of this Lab Report System are:

- All invalid data and inconsistent data must be detected as soon as that data is entered (interactive).
- Handle two kinds of inconsistency errors:
  1. A 'Real' error. For example, a male patient with a lab report 'pregnant.' In this case, the end-user should reenter the correct input data.
  2. A 'Suspicious' error. For example, an infant with a lab report 'arthritis.' In this case, the system should request the end-user either to verify or to reenter the data.

Solution

Write an SCL program behind an FSEDIT screen that performs the validation where each new report is a new observation in the data set.

For the first kind of inconsistency error, we use the SCL statements ERRORON and ERROROFF to flag the error.

For the second kind of inconsistency error, we pop-up a verification field. We use the SCL statements PROTECT and UNPROTECT. It will appear as if the question field is created as soon as it is needed.

Screen

Command = = = >

* XYZ Hospital Lab Report System *

Patient Information:
Name: &NAME
Sex: &SEX
Age: &AGE
Diagnosis: &RESULT
&dummy &VERIFY

Program
init:
  _msg_ = 'Please enter your data';
  verify = _blank_
  dummy = _blank_
  protect verify;
  return;

main:
  _msg_ = 'Data is accepted';
  dummy = _blank_
  link turnoff;
  if verify = 'Y' then
    link checkit;
    else
      verify = _blank_
      return;
  term:
    return;

turnoff:
  erroroff all;
  dummy = _blank_
  protect verify;
  return;

checkit:
  select (result);
  when('pregnant')
    /* a real inconsistent error */
    if sex = 'M' then
      do;
      erroron sex result;
      cursor sex;
      _msg_ = 'Error: inconsistent values entered';
      end;
  when('diaper rash')
    if age > 10 then
      link susprare;
    when('arthritis')
      if age < 15 then
        link susprare;
    otherwise;
    end;
  return;

susprare:
  _msg_ = 'Warning: Suspicious data. It is extremely rare for a ' age result cursor verify;
  erroron age result;
  cursor verify;
  unprotect verify;
  dummy = 'Verify the inconsistent data (Y/N)';
  return;

1160