CHAPTER 7 Using Other SAS Software Products

Introduction

SCL provides many of the same features as the base SAS language. However, some SCL features differ slightly in functionality from base SAS language features. Also, although SCL provides a rich set of features, it does not provide functions and statements to accomplish directly all of the data access, management, presentation, and analysis tasks that SAS software can perform, nor can it provide the equivalent for every command that is available under your host operating system. However, SCL does provide the following features:

- the SUBMIT statement, which provides access to other features of SAS software by generating SAS statements and then submitting them to SAS software for processing.
- the SYSTEM function, which provides access to host operating systems by issuing host operating system commands.

Using SAS DATA Step Features in SCL

SCL supports the syntax of the SAS DATA step with the exceptions and additions noted. Refer to *SAS Language Reference: Dictionary* for details about the SAS language elements that are available in the DATA step.

SCL does not support the DATA step statements that relate specifically to creating SAS data tables, such as the DATA, SET, INFILE, and DATALINES statements. However, SCL does provide special functions that can perform equivalent SAS table manipulations. See Chapter 11, "Using SAS Tables," on page 165 for details.

Statements

"SCL Language Elements by Category" on page 235 lists the statements that are supported by SCL and tells you where they are documented. The ARRAY, DO, LENGTH, PUT, and SELECT statements are different in SCL. The differences are documented in their entries in Chapter 16, "SAS Component Language Dictionary," on page 249. The following list shows the DATA step statements that are valid in SCL programs and notes differences between a statement's support in SCL and in the DATA step.

ARRAY (Explicit)
: defines the elements of an explicit array. _NUMERIC_, _CHARACTER_, and _ALL_ are not supported.

assignment
: assigns values to variables.

comment
: documents the purpose of a program.

CONTINUE
: stops the processing of the current DO loop and resumes with the next iteration of that DO loop. See the dictionary entries for DO as well as CONTINUE for information about the differences in the behavior of this statement in SCL.

DO, iterative DO, DO-UNTIL, DO-WHILE
: repetitively execute one or more statements. SCL does not support the DO-list form of the DO statement, but it does support LEAVE and CONTINUE statements that extend the capabilities of DO-group processing.

END
: designates the end of a DO group or SELECT group.

GOTO
: jumps to a specified program label.

IF-THEN-ELSE
: enables conditional execution of one or more statements.

%INCLUDE
: accesses statements (usually from an external file) and adds them to the program when the SCL program compiles.

LEAVE
: stops executing the current DO group and resumes with the next sequential statement. See the dictionary entries for DO as well as LEAVE for information about the differences in the behavior of this statement in SCL.

LENGTH
: allocates storage space for character and numeric variables. In SCL, the LENGTH statement can set only the lengths of nonwindow variables.

LINK
: jumps to a specified program label but allows a return to the following statement. SCL allows nesting of up to 25 LINK statements.

NULL
: is an executable statement that contains a semicolon (;) and acts as a place holder.

PUT
: writes text to the LOG window.

RETURN
: returns control or a value to the calling routine or application. In SCL, RETURN can also return a value from the execution of a method.

RUN
: is an alias for the RETURN statement.

SELECT-WHEN
: enables conditional execution of one or several statements or groups of statements.

STOP
: is an alias for the RETURN statement.

SUM
: adds the result of an expression to an accumulator variable.

Functions

SCL supports all DATA step functions except LAG and DIF. See Table 15.1 on page 235 for a list of the DATA step functions that are supported by SCL. See *SAS Language Reference: Dictionary* for details about other DATA step functions that are supported by SCL.

Variables

Variables in SCL programs share most of the characteristics of variables in the DATA step such as default length and type. However, you should be aware of the differences described in the following sections. In addition, SCL variables can be declared to be local in scope to a DO or SELECT block.

Numeric Variables

A variable is assigned the numeric data type if its data type is not explicitly declared.

Character Variables

In SCL, the length of a character variable is determined as follows:

- For window variables, the maximum length of a variable is equal to the length of the corresponding control or field in the window.
- For character-type nonwindow variables, the length is 200 characters unless a different length is explicitly declared. However, you can use the DECLARE or LENGTH statement to change the length from a minimum of 1 character to a maximum of 32K characters. The maximum length of a nonwindow variable is not affected by the length of a string that is assigned to the variable in the SCL program. For example, suppose your SCL program contains the following statement and that the window for the application does not include a field named LongWord:

  ```
  LongWord='Thisisaverylongword';
  ```

  As a result of this assignment statement, SCL creates a nonwindow variable named LongWord with a maximum length of 200 characters. The length of the string in the assignment statement has no effect on the maximum length of the variable. By contrast, this same assignment in a DATA step would create a variable with a maximum length of 19 characters.

  As in the DATA step, the LENGTH function in SCL returns the current trimmed length of a string (the position of the nonblank character at the right end of the variable value). However, SCL also provides the MLENGTH function, which returns the maximum length of a character variable, as well as the LENGTH function with the NOTRIM option, which returns the untrimmed length of a string.

Expressions

SCL supports the standard DATA step expressions in an identical manner. The only exception is the IN operator, which has the following syntax:

i=variable IN (*list-of-values*) | *array-name*;

In SCL, the IN operator returns the index of the element if a match is found, or it returns 0 if no match is found. However, in the DATA step, the IN operator returns 1 if a match is found and 0 if no match is found. The IN operator is valid for both numeric and character lists as well as for arrays. If a list that is used with the IN operator contains values with mixed data types, then those values are converted to the data type of the first value in the list when the program is compiled.

In the following example, the statements using the IN operator are equivalent:

```
array list{3}$ ('cat','bird','dog');
i='dog' in ('cat','bird','dog');
i='dog' in list;
```

In SCL, this example produces I=3, whereas in the DATA step the example produces I=1. Also, the DATA step does not support the form **`i='dog' in list`**.

Submitting SAS Statements and SQL Statements

SCL programs can submit statements to execute both DATA steps and all the procedures in any product in SAS software. SCL programs can also submit Structured Query Language (SQL) statements directly to SAS software's SQL processor without submitting a PROC SQL statement. SQL statements enable you to query the contents of SAS files and to create and manipulate SAS tables and SAS views. SCL programs also enable you to submit command line commands to the Program Editor window for processing. Finally, SCL programs can submit statements for processing on your local host or on a remote host, if SAS/CONNECT software is installed at your site.

Submitting Statements Compared to Using SCL Features

You should submit statements when the task you want to perform is difficult or impossible using SCL features alone. Whenever equivalent SCL features are available, it is more efficient to use them than to submit SAS statements. For example, the following two sets of statements produce the same result, opening an FSEDIT window to display the SAS data table WORK.DATA1 for editing:

```
   /* This uses the SCL Feature. */
call fsedit('work.data1');


   /* This uses submitted statements. */
submit continue;
   proc fsedit data=work.data1;
   run;
endsubmit;
```

From within an application, fewer computer resources are required to execute the CALL routine in SCL than to submit statements to SAS software. Thus, the CALL routine is a better choice unless you need features of the procedure that the CALL routine does not provide (for example, the VAR statement in PROC FSEDIT to select which variables to display to the user).

Designating Submit Blocks

In SCL programs, you designate statements to be submitted to SAS software for processing by placing them in submit blocks. A *submit block* begins with a SUBMIT statement, ends with an ENDSUBMIT statement, and consists of all the statements in between. The following statements illustrate these characteristics:

```
SUBMIT;  ❶
   proc print data=work.data1;  ❷
      var a b c;     ❷
   run;
endsubmit;   ❸
```

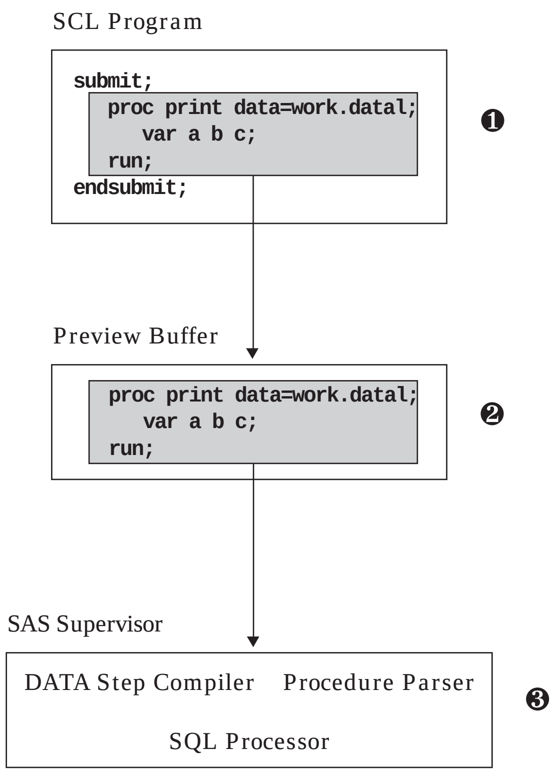
1 The SUBMIT statement starts the submit block.
2 These statements are submitted to SAS software when the program executes.
3 The ENDSUBMIT statement ends the submit block.

For details, see “SUBMIT” on page 676.

How Submit Blocks Are Processed

Figure 7.1 on page 82 illustrates how submit blocks are processed when they are executed (not when they are compiled). Submit blocks are not processed when you test a SAS/AF application with the TESTAF command.

Figure 7.1 Default Processing of Submit Blocks



```
submit;
  proc print data=work.datal;
     var a b c;
  run;
endsubmit;
```

```
  proc print data=work.datal;
     var a b c;
  run;
```

1 All of the code between a SUBMIT statement and the next ENDSUBMIT statement is copied into a special storage area called the *preview buffer*. The submitted code is not checked for errors when it is copied to the preview buffer. Errors in the submitted code are not detected until the statements or commands are executed.

2 The text in the preview buffer is scanned, and any requested substitutions are made. Substitution is discussed in “Substituting Text in Submit Blocks” on page 85.

3 The contents of the preview buffer are submitted to SAS software for execution. You can specify options to change where and when the contents of the preview buffer are submitted and to specify the actions that the SCL program takes after the statements are submitted. See “Modifying the Behavior of Submit Blocks” on page 83 for details.

Note: By default, code is not submitted immediately when the submit block is encountered in an executing program. Also, when a nested entry (that is, an entry that is called by another entry in the application) contains a submit block, the submitted code is not executed until the calling task ends, or until another submit block with a CONTINUE or IMMEDIATE option is encountered. Simply ending the entry that contains the submit block does not process submitted code. △

How Submitted Statements Are Formatted

By default, SCL reformats the submitted code when it copies it to the preview buffer. To conserve space, all leading and trailing spaces in the submitted text are removed. Semicolons in the submitted statements cause line breaks in the submitted text.

In some situations (for example, when the submitted code includes lines of raw data), you may want to prevent this formatting by SCL. You can do this by using a CONTROL statement with the ASIS option. When an SCL program contains a CONTROL ASIS statement, SCL honors the indention and spacing that appears in the submit block. Programs that use CONTROL ASIS are more efficient because the time spent on formatting is reduced. A CONTROL NOASIS statement restores the default behavior.

Modifying the Behavior of Submit Blocks

You can modify the default processing of submit blocks by specifying options in the SUBMIT statement. SUBMIT statement options control the following behaviors:

- □ when the code in the preview buffer is submitted for execution
- □ when the submitted code is processed and what happens after the submitted code is executed
- □ whether the submitted code is executed in the local SAS session or in a remote SAS session.

Controlling Where Submitted Code Is Executed

By default, code that is collected in the preview buffer using SUBMIT blocks is sent to SAS software for execution. SCL provides options for the SUBMIT statement that alter the default behavior. If you specify the CONTINUE option in the SUBMIT statement, you can control where code is submitted with the following options:

COMMAND
: submits the code in the preview buffer to the command line of the next window that is displayed. The code should contain valid commands for that window; otherwise, either errors are reported or the submitted commands are ignored.

EDIT
: sends the code in the preview buffer to the Program Editor window. You can modify your code in the Program Editor window and then submit it for execution.

SQL
: submits the code in the preview buffer to SAS software's SQL processor, from both TESTAF and AF modes. The SUBMIT SQL option enables you to submit the SQL statements without having to specify a PROC SQL statement. Submitting SQL statements directly to the SQL processor is more efficient than submitting PROC SQL statements.

Controlling What Happens After a Submit Block Executes

SCL also provides SUBMIT statement options that you can use to control what action, if any, the application takes after a submit block executes. These options are CONTINUE, IMMEDIATE, PRIMARY, and TERMINATE. Without one of these options, the code in a submit block is simply passed to the preview buffer, the application continues executing, and the code in the submit block is not processed by SAS software until the application ends.

CONTINUE
: suspends program execution while the submit block executes and then continues program execution at the statement that immediately follows the ENDSUBMIT statement. (CONTINUE is the only SUBMIT option that is valid in FSEDIT and FSBROWSE applications.)

IMMEDIATE
: stops program execution after the generated statements are submitted. Use this option with caution. Using this option in a labeled section that is executed individually when a CONTROL LABEL statement is in effect can prevent the execution of other labeled sections. A program in a FRAME entry does not compile if it contains a SUBMIT IMMEDIATE statement.

PRIMARY
: returns the program to the application's initial window after the generated statements are submitted. This option is useful when you want all the intermediate windows to close and you want control to return to a primary window in the current execution stream. This option causes looping if the current program is the primary window.

TERMINATE
: stops the SAS/AF task after the statements in the submit block are processed. This option is useful when an application does not need to interact with users after the submitted statements are processed. However, use TERMINATE with caution because re-invoking the application can be time-consuming.

Using SUBMIT CONTINUE in FSEDIT Applications

The behavior of a SUBMIT CONTINUE block in an FSEDIT application depends on how the application was invoked.

- □ If you invoked the application with a PROC FSEDIT statement, then the statements in the submit block cannot be processed until the FSEDIT session ends, even when you specify SUBMIT CONTINUE. The statements cannot be executed as long as the FSEDIT procedure is executing.
- □ If you invoked the application with an FSEDIT command or with a CALL FSEDIT routine from another SCL program, then the statements in the submit block can execute immediately as long as no other procedure is currently executing.

Submitting Statements to a Remote Host

By default, statements in a submit block are executed for processing on the local host. If SAS/CONNECT software is available at your site, you can also submit statements for processing on a remote host. To send submitted statements to a remote host, use the following form of the SUBMIT statement:

```
submit remote;
...SAS or SQL statements to execute
on a remote host...
endsubmit;
```

In situations where an application user can switch between a remote host and the local host, the user can issue the REMOTE command to force all submits to be sent to a remote host. The syntax of the REMOTE command is REMOTE <ON|OFF>. If neither ON nor OFF is specified, the command acts like a toggle.

The REMOTE option in the SUBMIT block takes precedence over a REMOTE command that is issued by an application user. A SAS/AF application must have a display window in order to issue and recognize the REMOTE command. Before SCL submits the generated code for execution, it checks to see whether the user has issued the REMOTE ON command. If a user has issued the command, SCL checks to see whether the remote link is still active. If the remote link is active, SCL submits the code for execution. If the remote link is not active, SCL generates an error message and returns. The preview buffer is not cleared if the submit fails.

Substituting Text in Submit Blocks

In interactive applications, values for statements in a submit block may need to be determined by user input or program input in the application. An SCL feature that supports this requirement is the substitution of text in submit blocks, based on the values of fields or SCL variables.

How Values Are Substituted in Submit Blocks

SCL performs substitution in submit blocks according to the following rules:

- When SCL encounters a name that is prefixed with an ampersand (&) in a submit block, it checks to see whether that name is the name of an SCL variable. If it is, then SCL substitutes the value of that variable for the variable reference in the submit block. For example, suppose a submit block contains the following statement:

  ```
  proc print data=&table;
  ```

  If the application includes a variable named TABLE whose value is **`work.sample`**, then this statement is passed to the preview buffer:

  ```
  proc print data=work.sample;
  ```

- If the name that follows the ampersand does not match an SCL variable, then no substitution occurs. The name is passed unchanged (including the ampersand) with the submitted statements. When SAS software processes the statements, it attempts to resolve the name as a macro variable reference. SCL does not resolve macro variable references within submit blocks. For example, suppose a submit block contains the following statement:

  ```
  proc print data=&table;
  ```

  If there is no SCL variable named TABLE in the application, then the statement is passed unchanged to the preview buffer. SAS software attempts to resolve &TABLE as a macro reference when the statements are processed.

CAUTION:

Avoid using the same name for both an SCL variable and a macro variable that you want to use in submitted statements. SCL substitutes the value of the corresponding SCL variable for any name that begins with an ampersand. To guarantee that a name is passed as a macro variable reference in submitted statements, precede the name with two ampersands (for example, &&TABLE). △

Specifying Text for Substitutions

If an SCL variable that is used in a substitution contains a null value, then a blank is substituted for the reference in the submitted statements. This can cause problems if the substitution occurs in a statement that requires a value, so SCL allows you to define a replacement string for the variable. If the variable's value is not blank, the complete replacement string is substituted for the variable reference. To define a replacement string, you can use either the Replace attribute (for a control or field) or the REPLACE statement.

Using the REPLACE Statement

The REPLACE statement acts as an implicit IF-THEN statement that determines when to substitute a specified string in the submit block. Consider the following example:

```
replace table 'data=&table';
   ...more SCL statements...
submit;
   proc print &table;
   run;
endsubmit;
```

If the SCL variable TABLE contains `''`(or **`_BLANK_`**), then these statements are submitted:

```
proc print;
run;
```

If the SCL variable TABLE contains **`work.sample`**, then these statements are submitted:

```
proc print data=work.sample;
run;
```

Using the Replace Attribute

In SAS/AF applications, you can also can define replacement strings for a window variable using the Replace attribute in the properties window (for a control) or the attribute window (for a field). The text that you specify for the Replace attribute is substituted for the variable name when the variable name is preceded with an ampersand in submitted statements.

Issuing Commands to Host Operating Systems

SCL programs can use the SYSTEM function to issue commands to host operating systems. For example, an SCL program may need to issue commands to the operating system in order to perform system-specific data management or control tasks or to invoke non-SAS applications.

An SCL program can issue any command that is valid for the operating system under which an application runs. SCL places no restrictions on commands that are issued to an operating system, nor does SCL check command strings for validity before passing them to the operating system.

Using Macro Variables

Macro variables, which are part of the macro facility in base SAS software, can be used in SCL programs. Macro variables are independent of any particular SAS table, application, or window. The values of macro variables are available to all SAS software products for the duration of a SAS session. For details, refer to macro variables in *SAS Macro Language: Reference*. In SCL programs, you can

- store values in macro variables (for example, to pass information from the current SCL program to subsequent programs in the application, to subsequent applications, or to other parts of SAS software).
- retrieve values from macro variables (for example, to pass information to the current SCL program from programs that executed previously or from other parts of SAS software, or to pass values from one observation to another in FSEDIT applications).

Examples of types of information that you frequently need to pass between entries in an application include

- names of SAS tables to be opened
- names of external files to be opened
- identifiers of open SAS tables
- file identifiers of open external files
- the current date (instead of using date functions repeatedly)
- values to be repeated across rows in an FSEDIT session.

Storing and Retrieving Macro Variable Values

To assign a literal value to a macro variable in an SCL program, you can use the standard macro variable assignment statement, %LET. For example, the following statement assigns the literal value **sales** (not the value of an SCL variable named SALES) to a macro variable named DSNAME:

```
%let dsname=sales;
```

Macro variable assignments are evaluated when SCL programs are compiled, not when they are executed. Thus, the %LET statement is useful for assigning literal values at compile time. For example, you can use macro variables defined in this manner to store a value or block of text that is used repeatedly in a program. However, you must use a different approach if you want to store the value of an SCL variable in a macro variable while the SCL program executes (for example, to pass values between SCL programs).

Macro variables store only strings of text characters, so numeric values are stored as strings of text digits that represent numeric values. To store values so that they can be retrieved correctly, you must use the appropriate CALL routine. The following routines store the value of a macro when an SCL program runs:

CALL SYMPUT
: stores a character value in a macro variable.

CALL SYMPUTN
: stores a numeric value in a macro variable.

For example, the following CALL routine stores the value of the SCL variable SALES in the macro variable TABLE:

```
call symput('table',sales);
```

To retrieve the value of a macro variable in an SCL program, you can use a standard macro variable reference. In the following example, the value of the macro variable TABLE is substituted for the macro variable reference when the program is *compiled*:

```
dsn="&table";
```

The function that you use to retrieve the value of a macro variable determines how the macro variable value is interpreted. The following functions return the value of a macro variable when a program runs:

SYMGET
: interprets the value of a macro variable as a character value.

SYMGETN
: interprets the value of a macro variable as a numeric value.

Using the Same Name for Macro Variables and SCL Variables

Using the same name for a macro variable and an SCL variable in an SCL program does not cause a conflict. Macro variables are stored in SAS software's global symbol table, whereas SCL variables are stored in the SCL data vector (SDV). However, if your program uses submit blocks and you have both a macro variable and an SCL variable with the same name, then a reference with a single ampersand substitutes the SCL variable. To force the macro variable to be substituted, reference it with two ampersands (&&). The following example demonstrates using a reference that contains two ampersands:

```
dsname='sasuser.class';
call symput('dsname','sasuser.houses');
submit continue;
   proc print data=&dsname;
   run;
   proc print data=&&dsname;
   run;
endsubmit;
```

The program produces the following:

```
proc print data=sasuser.class;
run;
proc print data=sasuser.houses;
run;
```

Using Automatic Macro Variables

In addition to macro variables that you define in your programs, SAS software provides a number of predefined macro variables for every SAS session or process. These automatic macro variables supply information about the current SAS session or process and about the host operating system on which the SAS session is running. For example, you can use the automatic macro variable SYSSCP to obtain the name of the current operating system. Automatic macro variables are documented in *SAS Macro Language: Reference.*

When you use automatic macro variables, remember to use the appropriate routines and functions to set and retrieve variable values. For example, consider the following program statements. The first uses a macro variable reference:

```
jobid="&sysjobid";
```

The second uses an SCL function:

```
jobid=symget('sysjobid');
```

The macro variable reference, designated by the & (ampersand), is evaluated when the program is compiled. Thus, the identifier value for the job or process that compiles the program is assigned to the variable JOBID. Assuming that the preceding two statements were compiled by an earlier SAS process, if you want the JOBID variable to contain the identifier for the current process, then you must use the second form (without the &). The SYMGET function extracts the macro variable value from the global symbol table at execution.

Note: The values that are returned by SYSJOBID and other automatic macro variables depend on your host operating system. △

The correct bibliographic citation for this manual is as follows: SAS Institute Inc., *SAS® Component Language: Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999.

SAS® Component Language: Reference, Version 8

Copyright © 1999 by SAS Institute Inc., Cary, NC, USA.

ISBN 1–58025–495–0

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, October 1999