03/05/98

6. Comments and Metastatements

6.1 Overview

VPLX applications are organized into steps, and most VPLX statements have a meaning that depends on the step for context. This chapter discusses comments and metastatements that have the same meaning regardless of their placement in the command file. The metastatements and the two primary forms of comments may occur at the beginning of a command file, within a step, or between steps.

Section 6.2 discusses the two primary methods for inserting comments into the VPLX command file. These two methods, “!”, and “/*” paired with “*/”, have appeared extensively in the examples in the previous chapters. The section provides more precise rules about their use and their interaction. For completeness, Section 6.3 describes other recognized forms of comments, mostly to support legacy VPLX applications using older syntax.

Metastatements begin in position 1. The metalanguage is not required to accomplish any specific statistical function in VPLX; instead, it is available to simplify the writing of complex applications.

Section 6.4 describes IGNORE, which can be used to turn off the execution of a block of VPLX code and suppress its appearance in the print file. Section 6.5 describes ECHO, which can be used to control the listing of the command file in the print file.

The SET statement, presented in Section 6.6, permits a rudimentary macro capability. The use of SET with the statements ON and OFF follows in Section 6.7. Section 6.8 describes INCLUDE, which directs VPLX to read commands from another file.

6.2 Commenting With “!”, “/*”, and “*/”

In general, the appearance of “!” in a command line causes the remainder of the line to be treated as a comment. Numerous examples appear in the previous chapters. An exception is made for “!” within the scope of single apostrophes,

```
        class viewer (1/2) 'Yes' 'No' ;         ! This is a comment
        cat reaction (1/2) 'Excited!' 'Bored'; ! Comment starts here
```

The appearance of the comment does not by itself end a statement in the same manner as a semicolon. Statements can appear on multiple lines with accompanying comments.

The other recommended form of comment is enclosed in pairs "/*" and "*/", which may appear in a single line or span multiple lines. Appearance of "/*" introduces a comment regardless of "!".

```
! This is a comment.  /*
this entire line is a comment
  */
```

Unlike SAS, VPLX recognizes nesting of comment pairs. For example, two "/*"'s in succession must be balanced by two "*/"'s before the comment ends. This permits commenting a large section of code that may already have been commented.

```
/* beginning of comment
/* this line is an imbedded comment */
  the end of comments follows */
```

Although it is unlikely to be often needed, a special rule allows inclusion of either "/*" or "*/" within the scope of apostrophes to avoid treatment as the beginning of a comment. Either should be doubled to be translated into the desired pair.

```
cat symbols (1/2) '/ alone'
     '/*/* combination' ;
```

The second label would contain a single "/*".

6.3 Other Commenting Commands

VPLX will continue to support other forms of commenting in legacy applications, but each is less useful than the "/*" "*/" pair, which is a recent addition to VPLX. The first of these alternative forms is not permitted within a step employing the new syntax, the second is easily replaced by the "/*" "*/" pair, and the third has restricted usage. For completeness, these forms are given here but their further use is discouraged. This first was the original commenting form in VPLX.

```
comment   This form of commenting begins with comment (or just c)
          in position 1.  It may extend to an arbitrary number of lines
          and ends when another command appears in position 1.

          It may not be used within a step written in the new syntax
          because it would not have a clearly defined end.  If it does
          appear within the new syntax, VPLX will consider it to end
          the statements for the current step.
```

The second form uses a starting “longcomment” and ending “longcomment end”, allowing all lines in between to be treated as comments, regardless of whether individual lines have characters in position 1.

```
longcomment      ! beginning statement, starting in position 1

This form of commenting treats all intervening lines as comments
until the matching end statement.

longcomment  off ! ending statement, starting in position 1.
```

Of course, this function can be achieved by a “/*” and “*/” pair.

A third form can be used within the new syntax only and mirrors a form in SAS.

```
    * This form begins with a '*' beyond position 1, and
      extends until an ending ';' ;
```

Note that the first semicolon does not end the comment because it is within apostrophes. The author recommends “/*” and “*/” pairs in preference to this form, however, because the pairs represent a more salient marking.

6.4 IGNORE

IGNORE performs a feature similar to commenting but suppresses the echoing of the lines to the print file.

```
ignore;       ! beginning statement, starting in position 1

All intervening lines are not interpreted and also suppressed from the
print file.  This is useful when an existing block of code is to be
skipped.  Omitting such code from the print file reduces visual clutter.

ignore  off; ! ending statement, starting in position 1.
```

IGNORE statements can be nested, and an equal number of IGNORE OFF statements are required to balance the nesting (*1*). The “OFF” portion of an IGNORE OFF statement must appear in the same line as “IGNORE”.

For compatibility with the old VPLX syntax, the semicolon following IGNORE or IGNORE OFF is optional.

6.5 ECHO

ECHO controls printing of commands to the print file. The forms are:

- ECHO OFF - to eliminate almost all normal messages and echoing of commands to the print file. Statistical outputs, such as displays, are still shown.

- ECHO or ECHO ON - to resume messages and echoing of commands to the print file.

- ECHO LITE - to allow selective printing of messages and echoing of initial commands of steps, substitutions based on SET (Section 6.6), and INCLUDE (Section 6.8) statements. This option is intermediate in degree between the preceding two. Thus, the print file shows an outline of what was done, without the details.

Each form must begin in position 1. Regardless of the current status of ECHO, the ECHO statements themselves and fatal error messages are sent to the print file. For consistency with earlier syntax, the ending semicolon is optional.

ECHO affects printing only. ECHO has no effect on the execution of commands.

To illustrate, the 3-line command file, which uses the INCLUDE statement in Section 6.8,

```
!  i6-1.crd
echo lite;
include i1-2.crd;
```

results in the print file,

```
echo lite;

include i1-2.crd;

scratch1  temp1.tmp;          ! Optional naming of scratch files.
... etc.

create  in = i1-1.dat  out = vplx1.vpl ;  ! Beginning of CREATE step

    Stratified jackknife replication assumed

    Size of block   1  =              3

    Total size of tally matrix =      3

    End of primary input file after obs #       6
```

```
         3 strata observed on incoming file


display                                    ! Beginning of DISPLAY step

                                                 Estimate      Standard error

Sample N (wtd) for block   1                       6.0000               .0000

rooms                    :  MEAN                   6.0000               .2357

persons                  :  MEAN                   4.0000               .4082

rooms                    :  TOTAL                 36.0000              1.4142

persons                  :  TOTAL                 24.0000              2.4495
```

Exhibit 6.1 An extract from i6-1.lis, illustrating the effect of ECHO LITE. The ECHO and INCLUDE statements appear in the print file, as well as the SCRATCHx, CREATE, and DISPLAY statements, some diagnostics from the CREATE step, and the results of the DISPLAY.

A similar version with ECHO OFF produces,

```
echo off;
                                                 Estimate      Standard error

Sample N (wtd) for block   1                       6.0000               .0000

rooms                    :  MEAN                   6.0000               .2357

persons                  :  MEAN                   4.0000               .4082

rooms                    :  TOTAL                 36.0000              1.4142

persons                  :  TOTAL                 24.0000              2.4495
```

Exhibit 6.2 An extract from i6-2.lis, illustrating the effect of ECHO OFF. Only the results of the DISPLAY appear.

6.6 SET

6.6.1 General Description The SET command may be used at any point to assign a symbolic name to a string of characters. The form for the string name follows the same rules as for variables (*2*). SET must begin in position 1. The syntax is,

```
SET  stringname  =  set of characters;
```

Once `stringname` has been defined in this way, then VPLX will substitute the assigned set of characters in place of `&stringname&` on all subsequent occurrences in the command file.

For example, one might assign,

```
set  basedatafile = /data5/survey.dat;
set  workdirect  = /work/;
```

The subsequent appearance in the input command file of

```
create    in = &basedatafile&
          out = &workdirect&tally1.vpl;
```

will appear in the VPLX print file as,

```
create    in = &basedatafile&
create    in = /data5/survey.dat

          out = &workdirect&tally1.vpl;
          out = /work/tally1.vpl;
```

to show both the command lines as read and the resulting substitutions. (Line spacing is used in the print file, however, to emphasize the pairing of command line with resulting substitution.) VPLX then interprets the statements after the substitutions have been made.

SET statements may appear anywhere. A subsequent SET statement can reassign a new string to a previously assigned string name,

```
set   mmmyy = jan98;
include  monthtal.vsk;
set   mmmyy = feb98;
include  monthtal.vsk;
set   mmmyy = mar98;
include  monthtal.vsk;
```

where `monthtal.vsk` is a file employing `&mmmyy&` to define files or other information to process one month's worth of data.

The SET statement facilitates the design of general VPLX applications. Suppose, for example, that one wanted to run the same VPLX application on two different computer platforms. A VPLX command file could begin with SET statements to define each of the files or directories used by the application, and all subsequent referencing of files could be through the assigned string names.

Simply by changing the SET statements at the beginning, one could then use the application on a different data set or port the application to a different computer environment.

6.6.2 Apostrophes Normally, the length of the string is determined by the first nonblank character after the " = " in the SET statement, and the last nonblank character preceding the ending semicolon. To begin a string with one or more blanks, a single apostrophe must be used. In general, an initial apostrophe is never considered part of the string. Similarly, an ending apostrophe may be used to signal the end of the string, and is not treated as part of the string. Consequently, the next two lines from the command file,

```
  set  basedatafile = ''node01"name passwd"::survey.dat'' ;
  create   in = &basedatafile&
```

will be displayed on the VPLX print file as

```
  set  basedatafile = ''node01"name passwd"::survey.dat'' ;

  create   in = &basedatafile&
  create   in = 'node01"name passwd"::survey.dat'
```

which removes the outer set of apostrophes, leaving the inner pair. (Because the file name includes a blank, it must be enclosed in apostrophes (*3*).) Again, the print file shows both the original command line and the resulting substitution.

6.6.3 Rules on Substitution. Except for the range of IGNORE, Section 6.4, and MACROWRITE, Section x.xx, VPLX will treat `&`*`stringname`*`&` — an initial "&", a valid string name, and an ending "&" — as a request for a substitution based on a previously defined SET statement. If no match exists, VPLX treats the occurrence as a fatal error, unless it occurs within a comment or the range of IGNORE.

As it makes substitutions, VPLX will allow the incoming line to expand into a buffer of 240 characters, which it will then interpret. For example,

```
  reweight in = &datafile& out = &newrepwts& vplxin = &transout& ;
```

could easily be longer than 80 characters after substitution. As long as the substitutions do not result in a line of greater than 240 characters, VPLX will interpret the resulting line. The resulting line will be wrapped on multiple lines in the print file. For aesthetic reasons, it is preferable to avoid lines that extend beyond 80 characters after substitution, but an error will occur only if the limit of 240 is exceeded.

I.6.8

Concatenation of strings is allowed; the command statements

```
set   workdirect = /work/;
set   tallyfile = tally1.dat;
display   out = &workdirect&&tallyfile&
```

appear in the VPLX print file as,

```
set   workdirect = /work/;
set   tallyfile = tally1.dat;

display   out = &workdirect&&tallyfile&
display   out = /work/tally1.dat
```

If, however, VPLX encounters a " && " in which the first ampersand is not the end of a previous `&stringname&` as above, then VPLX will include a single " & " on the print file and not attempt a substitution. For example,

```
/*  &&tallyfile&& will be defined later */
```

appears in the print file as

```
/*  &&tallyfile&& will be defined later */
/*  &tallyfile& will be defined later */
```

without any attempt to substitute a string, whether or not `&tallyfile&` has been defined in a previous SET statement at that point.

Substitutions may occur within SET statements,

```
set  mmmyy  = jan91;
set  outfile = &mmmyy&cps1.dat;
display  out = &outfile&
```

which results in

```
set  mmmyy  = jan91;

set  outfile = &mmmyy&cps1.dat;
set  outfile = jan91cps1.dat;

display  out = &outfile&;
display  out = jan91cps1.dat;
```

6.6.4 Multiline Substitutions. A string name may be set to more than one line,

```
set newrepwts  = temp$:repweights.dat
     (lrecl = 1100);
reweight  in = datafile.dat
          out = &newrepwts& ;
```

results in the interpretation,

```
set newrepwts  = temp$:repweights.dat
     (lrecl = 1100);

reweight  in = datafile.dat

          out = &newrepwts& ;
          out = temp$:repweights.dat
 (lrecl = 1100) ;
```

In this example, VPLX added a blank before `(lrecl = 1100)`. Note that VPLX generally assumes that any continuation line of a SET statement is intended to be used as a continuation line and will place an initial blank as the second and subsequent lines. An initial apostrophe may be used to direct VPLX to do otherwise, however, as in,

```
set  scratchspec = 'scratch1  file1.tmp;'
      'scratch2  file2.tmp;'
      'scratch3  file3.tmp;'
      'scratch4  file4.tmp;'
      'scratch5  file5.tmp;'
```

Generally, multiline SET statements are primarily intended to accommodate file information, such as the use of (lrecl=1100) in the VMS environment. In general, SET is not well suited for representing multiple lines of VPLX instructions, in spite of the preceding example (*4*). A preferable means to represent multiple lines of VPLX code is to use INCLUDE of Section 6.8.

6.7 OFF and ON

OFF and ON are metacommands designed to work with SET. The command OFF is a synonym for IGNORE. The line OFF OFF, is equivalent to IGNORE OFF. Thus,

```
set  switch = off;
&switch& ;
One or more lines that will be ignored if switch = off
&switch& off ;
```

ON is not the direct opposite of IGNORE, that is, it does not stand for IGNORE OFF. Instead, it is effectively a 1-line comment, that is, it is not further interpreted by VPLX. Similarly, ON OFF also has the same force as a comment, rather than having the meaning IGNORE. If ON appears within the scope of a previous OFF or IGNORE, it does not change the effect of the OFF or IGNORE.

The asymmetric meaning of OFF and ON serves a purpose. When used with SET, OFF and ON provide a simple means to identify blocks of code that can be turned off or on by a SET statement. To illustrate,

```
set  totalvar = on;
set  withinvar = off;
...
&totalvar&;
one or more lines of VPLX code to be executed only for
   computing total variance
&totalvar&  off;
&withinvar&;
one or more lines of VPLX code to be executed only for
   computing within variance (conditional on the selection
   of primary sampling units)
&withinvar&  off;
```

would execute the lines of code for total variance but ignore those for within variance. When VPLX encounters `&totalvar&` in the example, it translates it to `on` as a result of the preceding SET statement, simply ignores it as a comment, and interprets the command lines for total variance. When VPLX reaches `&totalvar&  off`, the translation to `on  off` is similarly treated as a 1-line comment. On the other hand, `&withinvar&` and `&withinvar& off` have the same effect as `ignore` and `ignore  off`, causing VPLX to ignore the command lines for within variance. Simply by changing the two SET statements, however, one could compute within variance instead. Thus, complex VPLX applications may be written where blocks of commands are controlled by set instructions at the beginning of the command file.

6.8 INCLUDE

6.8.1 General Description of INCLUDE: Section 6.3 presented an example of INCLUDE,

```
echo lite;
include i1-2.crd;
```

INCLUDE directs VPLX to read from another file as the command file. The syntax is

```
include    filename ;
```

where INCLUDE begins in position 1. VPLX continues reading commands from the new file until it reaches the end of file. At that point, control then returns to the file in which the INCLUDE statement appeared.

VPLX assumes that the INCLUDEd file contains commands in the same form as the primary command file. In particular, VPLX interprets only positions 1-80 from any INCLUDEd file.

6.8.2 Nesting of INCLUDE Statements INCLUDE may point to a file that itself has one or more INCLUDE statements. The process becomes analogous to a stack of plates. Each INCLUDE statement adds another plate to the stack, which is removed each time VPLX finishes reading the file to the end. VPLX is sensitive to the depth of the stack: an INCLUDE in the initial file may point to a first alternate file, which may point to a second alternate file, which may point to a third, *etc*.

There is a maximum possible depth of stack (*5*) of approximately 7, depending on whether VPLX itself is making partial use of the stack in parsing CREATE, REWEIGHT, and some other steps. VPLX monitors the maximum depth of stack and will terminate if resources are exceeded; the author is unaware of any practical application actually requiring more than the current allowed depth. Since every time an end-of-file is reached on an INCLUDEd file the stack is reduced by one, a VPLX application may contain an arbitrarily large number of INCLUDE statements as long as the maximum depth does not exceed the limit at any one point.

Example i6-3 illustrates the nesting of INCLUDE. For clarity, the command file is first shown in the following exhibit

```
!  i6-3.crd
/* file i6-3a.crd contains the next 4 lines:
set a = 1;                    ! line 1 of i6-3a.crd
include i6-3b.crd;            ! line 2 of i6-3a.crd
set e = 5;                    ! line 3 of i6-3a.crd
include i6-3d.crd;            ! line 4 of i6-3a.crd

file i6-3b.crd contains the next 2 lines:
set b = 2;                    ! line 1 of i6-3b.crd
include i6-3c.crd;            ! line 2 of i6-3b.crd

file i6-3c.crd contains the next 2 lines:
set c = 3;                    ! line 1 of i6-3c.crd
set d = 4;                    ! line 2 of i6-3c.crd

file i6-3d.crd contains the next line:
set f = 6;                    ! line 1 of i6-3d.crd

*/
```

```
include i6-3a.crd;
! a = &a&  b = &b&  c = &c&  d = &d&  e = &e&  f = &f&
```

Exhibit 6.3 The contents of i6-3.crd. The comments describe the contents of other files used by the example. After the comments, an INCLUDE statement directs reading from i6-3a.crd. The last line uses the string names defined by the SET statements in the INCLUDEd files.

The resulting print file is

```
                             VPLX  -  Version 1998.02
!  i6-3.crd
/* file i6-3a.crd contains the next 4 lines:
set a = 1;                    ! line 1 of i6-3a.crd
include i6-3b.crd;            ! line 2 of i6-3a.crd
set e = 5;                    ! line 3 of i6-3a.crd
include i6-3d.crd;            ! line 4 of i6-3a.crd

file i6-3b.crd contains the next 2 lines:
set b = 2;                    ! line 1 of i6-3b.crd
include i6-3c.crd;            ! line 2 of i6-3b.crd

file i6-3c.crd contains the next 2 lines:
set c = 3;                    ! line 1 of i6-3c.crd
set d = 4;                    ! line 2 of i6-3c.crd

file i6-3d.crd contains the next line:
set f = 6;                    ! line 1 of i6-3d.crd

*/


include i6-3a.crd;
    Assigned to unit 19

set a = 1;                    ! line 1 of i6-3a.crd

include i6-3b.crd;            ! line 2 of i6-3a.crd
    Assigned to unit 18

set b = 2;                    ! line 1 of i6-3b.crd

include i6-3c.crd;            ! line 2 of i6-3b.crd
    Assigned to unit  9

set c = 3;                    ! line 1 of i6-3c.crd

set d = 4;                    ! line 2 of i6-3c.crd

set e = 5;                    ! line 3 of i6-3a.crd

include i6-3d.crd;            ! line 4 of i6-3a.crd
    Assigned to unit 18

set f = 6;                    ! line 1 of i6-3d.crd
```

```
! a = &a&  b = &b&  c = &c&  d = &d&  e = &e&  f = &f&
! a = 1  b = 2  c = 3  d = 4  e = 5  f = 6
```

Exhibit 6.4 An extract of i6-3.lis, the print file from the command file in Exhibit 6.3. Only the final lines have been removed. VPLX reports the Fortran unit number used to read each INCLUDE file. VPLX shows both the final line as read and the result of the substitutions.

6.8.3 Restrictions on INCLUDE Generally, INCLUDE brings in a file with a series of VPLX command lines. In a few cases, these lines may even represent parts of a multi-line statement. There are two important limitations on INCLUDE, however. The restrictions are to avoid ambiguity and overly complex program logic within the VPLX program itself.

One restriction is that, once a file specification is started, INCLUDE must not be used to provide the remaining part of the file specification. This is particularly at issue for the initial commands of steps. For example, the following approach is **not allowed**:

```
reweight   in =  testdata1.dat
include    outfname.crd;
```

where the file `outfname.crd` is

```
      out = rewt.dat;
```

An INCLUDEd file may contain the entire initial statement, however. The following 2-line file may be INCLUDEd:

```
reweight   in =  testdata1.dat
           out = rewt.dat;
```

In general, INCLUDE is designed for larger sections of code than a file name. The SET feature, in Section 6.6, is better suited for file specifications and other short sections of VPLX code.

```
set   outfilename = rewt.dat ;
reweight   in =  testdata1.dat
           out = &outfilename& ;
```

To support older applications using the old syntax, VPLX treats the ending semicolon of the INCLUDE statement as optional. The second restriction is that, if INCLUDE is used without the ending semicolon, the next line must not contain "(" as the first nonblank character. The practice of always ending the INCLUDE statement with a semicolon will avoid this issue.

6.8.4 Uses of INCLUDE INCLUDE is quite useful in a wide variety of contexts. To suggest just some of them:

- One may want to INCLUDE values calculated by another VPLX step or another system into the VPLX step.

- If an input data file has many variables, it may be useful to write an INPUT statement to another file and to INCLUDE the file each time it is needed. This approach assures consistence of the information from one step to the next.

- Particularly in the TRANSFORM step, one may want to repeat the same set of commands several times, for example, in carrying out an iterative calculation. The group of repeated statements may be placed in a file and the file INCLUDEd as many times as necessary.

- If a calculation involves a significant number of VPLX steps, it may be useful to divide the steps into separate files and create a master control file that INCLUDEs them. If some steps succeed and not others, one can simply change what files are INCLUDEd from the master control file and restart the run, taking advantage of the successful steps.

The MACROWRITE step, described in Section x.xx, permits the writing of new files directly from the command file. These files may then be subsequently INCLUDEd in the same run.

NOTES

1. Additional IGNORE OFF statements without a preceding matching IGNORE have no effect. VPLX accepts an IGNORE without a subsequent balancing IGNORE OFF as canceling all lines that follow. A special rule applies to INCLUDEd files, described in Section 6.8. At the end of an INCLUDEd file, an IGNORE statement that has not been balanced by an IGNORE OFF within the INCLUDEd file will be turned off upon return to the calling command file. This feature prevents the commands in one file from being IGNOREd on the basis of a statement in another file.

2. Recent versions permit up to 24 characters for string names, instead of the current 12 for variable names. The string names may also incorporate periods, a feature not yet allowed for variable names.

3. The apostrophes are required to indicate the presence of a space in the filename. In this example, the space occurs between the name and password used to access a file across nodes. In general, however, bounding apostrophes are not required when one or more spaces are present in the character string.

4. One reason for not using set to create multiple line substitutions is that the substitution strings are stored in an array as data. It is possible to exceed the allotted space. Use of MACROWRITE and INCLUDE, on the other hand, stores information in external files, and is limited only by available space on the external file systems.

5. If practical examples are found requiring a deeper stack, then changes to PARAMETER statements in the Fortran source of VPLX could achieve this. Because this change would require system resources that could be used for other purposes, however, the current limit represents the best guess of what might be required by any practical application.