User-written procedures are an integral part of the SAS system. User-written procedures allow extension and customization of the SAS system to meet specific needs not already met by the current capabilities of the SAS system. Historically the languages available to program user-written procedures have been FORTRAN and PL/I. With the introduction of version 6 on the PC, the whole SAS system is being re-written in C. In order to develop version 6 on IBM mainframes SAS Institute developed a version of the LATTICE C compiler for IBM mainframes. With the SAS/C™ Native Compiler for IBM 370 systems, version 5 user-written procedures can be written in the C language.

There are several other significant enhancements to user-written procedures in version 5. The specification and parsing of a user-written procedure grammar has become simpler with the addition of a high-level language grammar parser. In 82.4 a user-written procedure grammar parser consisted of either Institute supplied assembler macros or the user's own assembler code. Beginning with version 5 the parser interface has been reworked so that the use of C, PL/I, or assembler languages to perform parsing is supported. Also the available non-custom parsing constructs have been greatly expanded. Two examples of the new functionality of non-custom parsing constructs are the RESTRICT statement of PROC REG and the PARMS statement of PROC NLIN. Examples of the PROC REG RESTRICT statement and the grammar used to parse the RESTRICT statement is given below.

```
RESTRICT 3*X4 = X2 + .25*11 - .5*X3;
RESTRICT X4 = X2 = X1 = -3*X3;
```

```
RESTSTMT =
  @STMTINIT(5)
  @SPECIAL(14)
  @LABEL(3)
  @STMTLIST(4,1)
  @STMTLIST(5,4)
  @STMTLIST(1,1)
"RESTR!CT"
  (LINCOMB
   ;@MARK(5)
   <"/" RESTOPT>>
  )
@MARK(5)
RESTOPT = "PRINT"
  @STMTEND
  @OPT(1),
```

The semantic actions @LISTVAL, @MARK, @SPECIAL are used to create two lists. The LIST 1 list consists of variable numbers delimited by negative markers. The markers indicate the type of coefficient associated with the corresponding variable. The LIST 2 list consists of the coefficients in floating point form. The lists LIST 4 and LIST 5 are initialized for later use by PROC REG. LIST 4 and LIST 5 will contain all of the information used to impose linear constraints. The construction of multiple constraints as shown below

```
equation = equation = equation = equation
```

is made possible by using the ; operator which allows constructs of the form

- A
- ABA
- AABAA
- etc.

An example of the PROC NLIN PARMS statement and the grammar used to parse the PARMS statement is as follows.

```
PARMS A=1 TO 10 BY .1 B=-2 C=1,3,4,5;
```

```
PARMSTMT =
  @STMTINIT(3)
  @SPECIAL(114)
  "PARAMETERS"
  @HAILE
  "="
  @LISTINISIT(3,4)
  @LISTCT(2,3) )
+ @SPECIAL(14)
  @STMTEND
```

SAS is a registered trademark of SAS Institute Inc., Cary, NC, USA. SAS/C is a trademark of SAS Institute Inc.
The NLIN PARMS statement generates three lists. LIST 1 contains the parameter names. LIST 3 contains a list of all initial values of the parameters. The @LISTCT semantic action places in LIST 2 the current number of elements in LIST 3. This allows PROC NLIN to determine for each of the parameters a list of starting values.

Another enhancement available to version 5 user-written procedure writers is the addition of printing interface routines. The use of the Institute supplied XP-routines allows user-written procedures to run under the SAS Display Manager. The XP routines allow user-written procedures in version 5 several new capabilities or easier implementation of output coding.

- Highlight table header information
- Use formats in printing
- Run under the SAS Display Manager
- Print log messages with substitution
- Easy composition of output line
- Use multiple SAS data sets without resetting current data set
- Use multiple Utility data sets without resetting current data set
- $PRL routines for printing column output
- $PUTMAT, $PUTMTL for printing matrices
- $PUTVEC for printing vectors

An example of the code used to set up printed column output using $PRL routines is given below.

```
/*----------PLAN FORMATING------------------------*/
CALL $PRLI(16,PRLPTR);
CALL $PRLD(PRLPTR,'# #VARIABLE DF#', 0, 'C', 12, 0, ADDR(PNAME), 0);
CALL $PRLD(PRLPTR,'# PARAMETER ESTIMATE#', 0, 'F', 12, 98, ADDR(BETA), 0);
CALL $PRLD(PRLPTR,'# STANDARD ERROR#', 0, 'F', 12, 98, ADDR(STD), 0);
CALL $PRLD(PRLPTR,'# T FOR HO: PARAMETER=0#', 0, 'F', 12, 3, ADDR(TRATIO), 0);
CALL $PRLD(PRLPTR,'# PROB > ITI#', 0, 'F', 12, 4, ADDR(PROBT), 0);
IF $SS1 THEN
  CALL $PRLD(PRLPTR, '# # TYPE I SS#', 0, 'F', 12, 98, ADDR(SSS1), 0);
END;
```

The above code produces the following output.

```
<table>
<thead>
<tr>
<th>Variable</th>
<th>DF</th>
<th>Prob &gt;</th>
<th>INTERCEP</th>
<th>1</th>
<th>0.2990</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X1</td>
<td></td>
<td></td>
<td>1</td>
<td>0.5107</td>
</tr>
<tr>
<td></td>
<td>X2</td>
<td></td>
<td></td>
<td>1</td>
<td>0.5214</td>
</tr>
<tr>
<td></td>
<td>X3</td>
<td></td>
<td></td>
<td>1</td>
<td>0.8956</td>
</tr>
</tbody>
</table>
```

The $PUTMAT routine can be used to print matrices stored in symmetric form, or in rectangular form stored by row or column. The routine also allows for printing of a submatrix of a matrix. Optional row and column headers can be specified as well as a title and/or a corner title. The code used to print a correlation matrix stored in symmetric form is given below.

```
CALL $PUTMAT(ZPZPTR,l,NX,1,HX,O,'SYM',12,'B',1,MODNAME,MODNAME,'CORRB', 'CORRELATION OF ESTIMATES');
```

The use of formats to produce user formatted output can be accomplished by using the XVLIST, XPAR, XVNAME, XPF and XPAR routine. Suppose that a single id variable can be specified by the user. The grammar might include the following production.

```
IDSTMT = ID |STMTPROC |STMTLIST(7,3)
```

The variable specified on the ID statement can be numeric or character. If the user-written procedure is invoked as below,

```
PROC USERWRITTEN;
  FORMAT IDVAR DATE.;
  ID IDVAR;
```

the following code shows how to place the formatted value of IDVAR in the output line.

```
FIELDPTR = ADDR(STMTFLD(7));
CALL XVLIST(INFILEID,7,FIELDPTR,1);
CALL XPAR(INFILEID,'F',1);
CALL XPAR(INFILEID,'I',1,1,NAMEPTR);
IDTYPE = NTYPE;
IDCODE = NCODE;
IDLEN = NLEN;
```
IDDEC = IFD;
IDFORM = IDCODE + IDDEC;

IF IDTYPE=2 THEN DO;
IF LINESLEFT<15 THEN CALL XPAGE;
IF PRINTBIT THEN DO;
/*----------PRINT STATISTICS OF FIT---------------*/
CALL XPCS(IDCVALUE,IDLEN,IDCODE,LINEPOS+1,LINEOUT);
END;
ELSE CALL XPF(IDVALUE,IDLEN,IDFORM,0);

Note for a character variable, LINEPOS must be incremented. The XPF call automatically increments LINEPOS.

The following code produces the Anova table of PROC REG.

IOFORM = IOCOOE + IDDEC;
IDDEC = NFD;
ELSE IF LINESLEFT<15 THEN CALL XPAGE;
ELSE IF PRINTBIT THEN DO;
	/*----------PRINT STATISTICS OF FIT---------------*/
CALL XPF(ANALYSIS OF VARIANCE',0,$CENTER(20));
CALL XPC('DEP VARIABLE: ',0,1);
CALL XPC(' SUM OF SQUARES',0,-7);
LINEINDENT=$CENTER(64);
END;
LINEINDENT='H';
CALL XPC('MODEL ',0,1);
CALL XPC('MEAN',0,-7);
CALL XPSKIP(2);
CALL XPF('ERROR ',0,1);
CALL XPF('MEAN',0,-7);
CALL XPSKIP(2);
CALL XPF('SOURCE OF SQUARES SQUARES',11,
	'F VALUE PROB>F',
	0,0);
CALL XPSKIP(2);
LINEINDENT='D';
CALL XPC('MODEL ',0,1);
CALL XPF('ADJ RSQ',0,0);
CALL XPF('CV',12,99,0);
CALL XPF('ROOT MSE ',0,5);
CALL XPF('MSR',12,98,-1);
CALL XPF('MSR',12,98,-1);
CALL XPF('FPROF',15,3,0);
CALL XPF('ADJRSQ',12,99,0);
CALL XPSKIP(1);
CALL XPF('SUM OVF',0,19);
CALL XPF('MEAN',0,0);
CALL XPF('ROOT MSE',0,0);
CALL XPF('SSR',12,98,-1);
CALL XPF('SSE',12,98,-1);
CALL XPF('DFE',5,0);
CALL XPF('DFRATIO',13,3,0);
CALL XPF('PROF',15,3,0);
CALL XPSKIP(1);
CALL XPF('SUM OF',0,19);
CALL XPF('MEAN',0,0);
CALL XPF('ROOT MSE',0,0);
CALL XPF('SSR',12,98,-1);
CALL XPF('SSE',12,98,-1);
CALL XPF('DFE',5,0);
CALL XPF('DFRATIO',13,3,0);
CALL XPF('PROF',15,3,0);
CALL XPSKIP(1);
CALL XPF('SUM OF',0,19);
CALL XPF('MEAN',0,0);
CALL XPF('ROOT MSE',0,0);
CALL XPF('SSR',12,98,-1);
CALL XPF('SSE',12,98,-1);
CALL XPF('DFE',5,0);
CALL XPF('DFRATIO',13,3,0);
CALL XPF('PROF',15,3,0);
CALL XPSKIP(1);
CALL XPF('SUM OF',0,19);
CALL XPF('MEAN',0,0);
CALL XPF('ROOT MSE',0,0);
CALL XPF('SSR',12,98,-1);
CALL XPF('SSE',12,98,-1);
CALL XPF('DFE',5,0);
CALL XPF('DFRATIO',13,3,0);
CALL XPF('PROF',15,3,0);
CALL XPSKIP(1);
CALL XPF('SUM OF',0,19);
CALL XPF('MEAN',0,0);
CALL XPF('ROOT MSE',0,0);
CALL XPF('SSR',12,98,-1);
CALL XPF('SSE',12,98,-1);
CALL XPF('DFE',5,0);
CALL XPF('DFRATIO',13,3,0);
CALL XPF('PROF',15,3,0);
CALL XPSKIP(1);
CALL XPF('SUM OF',0,19);
CALL XPF('MEAN',0,0);
CALL XPF('ROOT MSE',0,0);
CALL XPF('SSR',12,98,-1);
CALL XPF('SSE',12,98,-1);
CALL XPF('DFE',5,0);
CALL XPF('DFRATIO',13,3,0);
CALL XPF('PROF',15,3,0);
CALL XPSKIP(1);
ALL of the above examples are reasons SAS Institute is encouraging users to upgrade user-written procedures to version 5 standards.

User-written procedures written using the 82.4 interface will run under version 5 in non Display Manager mode. This allows user's time to convert to version 5 standards. In version 5 user-written procedures can be written in C, PLI, or FORTRAN. However only C and PLI procedures can be written using version 5 interface routines. If one needs to write a new FORTRAN user-written procedure then one must use 82.4 interface routines. There are several reasons for not providing a version 5 FORTRAN interface. First, there are several different FORTRAN compilers that would need to be supported. Those compilers may treat literal character constants differently when calling the SAS interface routines. Second, finding FORTRAN data types to match data types used in SAS data structures is nontrivial problem. Third, to properly support FORTRAN user-written procedures a tremendous amount of new interface routines would have to be designed and implemented for manipulating SAS data structures. This would involve having different interface routines for the FORTRAN language. We would prefer to have a common interface for all languages. If you are interested in writing 82.4 style user-written procedures in FORTRAN, please contact the Marketing Support Department. An alternate approach would be to write a version 5 PLI procedure shell around FORTRAN subroutines. It should be possible to build a C procedure shell around FORTRAN subroutines. I have not yet tested a C shell but I plan to try a test procedure after SUGI. If you are interested in writing a C shell around FORTRAN subroutines please contact me at SAS Institute for more information.

Both PLI and C user-written procedures enjoy full support under version 5. Of the two languages only C user-written procedure will be portable to the PC SAS System when recoded using version 6 interfaces. From my viewpoint C should be the language of choice if the programming expertise is available. The C language is more portable if properly coded and usually produces smaller and more efficient code. All of the above mentioned enhancements are reasons SAS Institute is encouraging users to convert existing user-written procedures to version 5 standards. With version 6 the user interface will definitely change. The version 5 interface will be much more similar to version 6 interface than the 82.4 interface. For more information on the difference between version 5 interfaces for C and for PLI refer to the proceedings from the advance tutorials section.

Now that I have talked about the status of version 5 user-written procedures, I would like to make some general comments about version 6 user-written procedures and the transition between version 5 and version 6. I will address the issue of transition first. Support of 82.4 and version 5 user-written procedures under version 6 is a difficult task. Well behaved version 5 user-written procedures should be portable under version 6 with few changes. However, support for 82.4 procedures that are ill-behaved may consist
of bringing version 5 up under version 6. For ill-behaved user-written procedures support may be possible only under MVS. We are currently studying the issue of supporting version 5 and 82.4 style procedures. If you have need of running 82.4 and version 5 style user-written procedures at the same time as running version 6 please contact the Marketing Support Department. Similarly the issue of supporting old style user-written data step functions and call routines is under study. It may be possible to support user-written data step functions and call routines as they exist. However, at this time the type of support that will be provided is unknown. If you have need of existing functions or routines under version 6 without recoding, please contact the Marketing Support Department.

Under version 6 user-written procedures and user written data step functions and call routines will be supported. Currently version 6 is being developed to be portable from the PC to other machines. Until version 6 is running on all machines, our internal interfaces are under going revisions. Once our internal interfaces are mature we will develop the version 6 user-written interface. One of our goals in developing the new user-written interface is to make it less sensitive to version changes. Part of the development of the version 6 user-written procedure interface is deciding what languages and compilers to support. Currently FORTRAN, PL/I and C are being considered. The languages and compilers chosen might vary from machine to machine. However C user-written procedures will be supported on all machines where user-written procedures can be supported. If you have need of a particular language other than C or a particular compiler please contact the Marketing Support Department.