RL-83-023

Science and Engineering Research Council
Rutherford Appleton Laboratory
CHILTON, DIDCOT, OXON, OX11 0QX

RL-83-023

XPLANT for ELECTRIC users

J C Gordon

March 1983

'The Science and Engineering Research Council does not accept any responsibility for loss or damage arising from the use of information contained in any of its reports or in any communication about its tests or investigations.'

RL-83-023

XPLANT for ELECTRIC users.

J C Gordon

Rutherford Appleton Laboratory
Chilton
Didcot
Oxfordshire
OX11 0QX

March 1983

FOREWORD

This document is not intended to be a complete technical description of the XPLANT[1] system. It is a simple introduction aimed at people familiar with the ELECTRIC editing system and the concepts of conditional editing , dynamic 'planting' of strings and 'supplying' of files. In this way it is hoped to facilitate the migration of users from ELECTRIC to CMS. It should be stressed that XPLANT is a powerful tool with many more facilities for conditional editing than ELECTRIC. This document will only describe a few of these to enable the experienced ELECTRIC users to transfer their work to CMS. It is hoped that after this introduction, the user will then exploit the greater power offered by XPLANT. The user is assumed to have a basic knowledge of CMS[2] as used at the Rutherford Appleton Laboratory[3] at the level of that given in the introductory guide[4] or on the introductory course given to new users of the system[5].

Section 1 is an introduction to the uses of XPLANT by the use of several annotated examples and comparison with similar ELECTRIC files. Section 2 contains a description of the XPLANT command and a few useful macros together with a description of the XEDIT macros which can be used to make conditional edits on a file. Section 3 lists the ELECTRIC delayed editing commands and suggests simple replacements when using XPLANT. Section 4 discusses using some of XPLANT's more powerful features to improve on the straightforward replacements suggested in section 3. For an overall introduction and comprehensive description of XPLANT the user is referred to Reference 1.

CONTENTS

NOTATION

The following conventions are used in this report.

(a) When a command, option or parameter is written in mixed case, the part written in upper case shows the shortest possible abbreviation.

(b) In the examples given, some lines are marked on the right by letters in parentheses (eg ....(a)), these letters refer to the corresponding points listed on the following pages.

(c) A list of items separated by | and enclosed by < > denotes a choice. Only one of the list should be given, without the < >.

(d) XPLANT labels and macro names are given in upper case for clarity. They can always be given in lower or mixed case.

(e) Whenever an example is given of a CMS session with a mixture of user input and replies from the system, the user input is emboldened.

1. INTRODUCTION

XPLANT is a program written by Systems Group at Rutherford Appleton Laboratory. Its purpose is to provide dynamic and stored editing facilities for CMS users in a way best suited to the VM/CMS and OS environments. The editing scheme is based on the previous PLANT program using special characters to introduce 'brackets' within the input file. All the text is in one file so it is easier to modify since existing edits can be seen and there is no system housekeeping to be done on a separate edit file.

The program performs a copy operation from a CMS file to the specified output medium which may be

(a) The user's terminal

(b) Another CMS file

(c) A job submitted to the MVT batch system, or the batch system at another VNET node.

As it performs the copy, XPLANT interprets edit strings (or 'brackets') in the file and outputs text resulting from the contents of these brackets. The selection of these edits can be controlled from the command line and the values of parameters within the file may be given either on the command line or in response to prompts from the file itself. The value of these parameters can be checked against a range of data-types and if they are not of the correct type a prompt may be issued for a correct value. The rest of this section shows a number of typical ELECTRIC files with their associated edit files and XPLANT files which perform equivalent tasks. These examples are in increasing order of complexity and the user who does not use all the power of ELECTRIC's delayed edits may wish to skip the later ones.

1.1 Example 1 - A Simple Job

Take as an example the ELECTRIC file shown below with its edit file.

```
FL=PYMAINDR.TESTJOB  NENT=  12,NBLK=  1
  1://US JOB (,,,),'TEST JOB'
  2://* JOB TO COPY CARDS TO DISK
  3://COPY EXEC GENER
  4://IN DD *
  5:C            A SHORT DATA FILE.
  6:     1     1  12.594    58.2379    42.1863    20.889     1563.4
  7:     1     2  5.239     45.259     73.1782    469.2      1269.47
  8:     2     3  12.4578    25.699    144.0      487.26     1300.00
  9://OUT DD  DSN=USER.USCOPY,VOL=REF=,DISP=NEW,
 10://    DCB=CARDS,SPACE=(TRK,)
 11:/*
EOP
```

```
FL=PYMAINDR.TESTJOB.ED  NENT=    4,NBLK=  1
 1 $P LN=    9.   1,C1= 15,C2= 26,CH=NO,DF=YS,NM=DSN
 1 $P LN=    9.   2,C1= 36,C2= 35,CH=NO,DF=NO,NM=VOL
 1 $P LN=   10.   1,C1= 10,C2= 14,CH=NO,DF=YS,NM=DCB
 1 $P LN=   10.   2,C1= 27,C2= 27,CH=NO,DF=NO,NM=TRACKS
EOP
```

The ELECTRIC command

```
EXEC TESTJOB,VOL=RHEL01,DSN='USER.USCARD',TRACKS=1
```

will submit the file to the designated batch system. The parameters VOL and DSN will be planted in the file. Note that some of the parameters (eg DCB) have default values which will be planted if they are not given on the command line.

A CMS file for processing by XPLANT to achieve a similar result would look as follows. The lower case letters in parentheses are references to the points on the following pages.

```
{$JOBCARD }                                                       ...(a)
//* JOB TO COPY CARDS TO DISK                                      ...(b)
//COPY EXEC GENER
//IN DD *
C           A SHORT DATA FILE.
    1    1  12.594    58.2379    42.1863    20.889     1563.4
    1    2  5.239     45.259     73.1782    469.2      1269.47
    2    3  12.4578    25.699    144.0      487.26     1300.00
//OUT DD VOL=REF={$P VOL},DSN={$P DSN USER.USCOPY},            ...(c)(d)
//  DISP=NEW,SPACE=(TRK,{TRACKS}),                                ...(e)
//   DCB=CARDS
/*
```

The corresponding command to submit this file to the MVT batch system is:-

```
XPLANT TEST (SUBMIT(VOL RHEL01 DSN USER.USCARD TRACKS 1    ...(f)
```

This will produce the following output on the user's terminal

```
PUN  011 DEFINED
Jobname "US" submitting to OS batch system.
PUN FILE 5805  TO  FEM      COPY 001   NOHOLD
PUN  011 DETACHED
R;
```

and the following file would be submitted to the MVT batch system.

```
//US JOB (1111,US,,,,,,,),
// 'VM/US/PH-37'
//* Command line:XPLANT TEST(SUB(VOL RHEL01 DSN USER.USCARD TRACKS 1
//* $JOBCARD called from "TEST      XPLANT    A1 1"
//* Submitted from RAL CMS userid "JCG" on 09/20/82 16:05:54
//* JOB TO COPY CARDS TO DISK
//COPY EXEC GENER
//IN DD *
C           A SHORT DATA FILE.
    1    1  12.594    58.2379    42.1863    20.889     1563.4
    1    2  5.239     45.259     73.1782    469.2      1269.47
    2    3  12.4578    25.699    144.0      487.26     1300.00
```

```
//OUT DD VOL=REF=RHEL01,DSN=USER.USCARD
//  DISP=NEW,SPACE=(TRK,1),
//   DCB=CARDS
/*
```

Note the following points.

(a) This is a call to the XPLANT $JOBCARD macro. When XPLANT encounters this call it places a HASP jobcard in the output stream. This jobcard uses the user's MVT id and account. Here all other HASP jobcard parameters assume their default values.

(b) All text that is not enclosed in brackets is copied to the output medium.

(c) The first bracket is a call to the $P macro. If the parameter VOL is given a value on the command line then this will be output in place of the bracket. If VOL is not given a value then $P will prompt for one and replace the bracket by the value given in response to the prompt.

(d) The second bracket on this line is another call to $P. This time the value of the parameter DSN will be output in place of the bracket, but if DSN does not have a value then the default value USER.USCOPY will be used.

(e) This bracket will be replaced by the value of the parameter TRACKS. If TRACKS has no value then the bracket will be removed and nothing output.

(f) The XPLANT command has a SUBMIT option to direct the output file to the MVT batch system.

This Example introduced the following XPLANT concepts.

(g) XPLANT processes special strings or 'brackets' which appear in the input file. In the output stream these brackets are replaced by text strings. The values of these strings depend on the contents of the brackets. All other text in the input file is copied directly to the output stream.

(h) XPLANT macros (whose names begin with $) are controlled by their argument lists which may themselves contain brackets. Macros may or may not produce output. (Examples $P,$JOBCARD)

(i) If a bracket does not call a macro then its contents are treated as an XPLANT parameter or 'label' and the value of this parameter is returned. The parameter may be assigned a value either on the command line or in a another bracket earlier in the file. If the parameter has no value then nothing will be placed in the output file in place of the bracket.

1.2 Example 2 - Checking Data Types

Below is a modified edit file for the ELECTRIC file shown above.

```
FL=PYMAINDR.TESTJOB.ED  NENT=   4,NBLK=  1
 1 $P LN=   9.  1,C1= 15,C2= 14,CH=DS,DF=YS,NM=DSN
 1 $P LN=   9.  2,C1= 24,C2= 23,CH=AN,DF=NO,NM=VOL
 1 $P LN=  10.  1,C1= 10,C2= 14,CH=NO,DF=YS,NM=DCB
 1 $P LN=  10.  2,C1= 27,C2= 27,CH=IN,DF=YS,NM=TRACKS
EOP
```

In this file the values of VOL, DSN and TRACKS are checked to be of types alphanumeric, OS dataset name and Numeric respectively.

If this file is submitted with the command line

```
PARM ID=XX,ACCT=1234,PRI=12,JOBNAME=USCOPY,LINES=1
EXEC TESTJOB,DSN=USER.XXDATA,VOL=RHEL01,TRACKS=1
```

ELECTRIC would submit the file to the MVT system with the appropriate values of the parameters given. If VOL, TRACKS or DSN do not have the specified type of value then the command would fail. ELECTRIC also recognises ID, ACCT, PRI, JOBNAME and LINES as parameter names and uses these values when submitting the job.

The equivalent XPLANT file would look like:-

```
{$JOBCARD }
//* JOB TO COPY CARDS TO DISK
//COPY EXEC GENER
//IN DD *
C          A SHORT DATA FILE.
    1     1  12.594     58.2379     42.1863     20.889      1563.4
    1     2  5.239      45.259      73.1782     469.2       1269.47
    2     3  12.4578     25.699     144.0       487.26      1300.00
//OUT DD VOL=REF={$P VOL *CHECK=VOL},DSN={$P DSN USER.USCOPY OSDSN},
//        DISP=NEW,SPACE=(TRK,{$P TRACKS 5 NUM}),
//        DCB=CARDS
/*
```

The Command line for XPLANT to submit this file to MVT would be too long to fit onto one line, so the user must use the PROMPT option. This option will prompt the user for parameters and he can enter as many as he likes in parameter-value pairs separated by a blank, one pair per line. The list is terminated by a null line. For Example , the command line

```
XPLANT TEST JOB(SUB PROMPT (ID XX ACCT 1234 PRI 12 TRACKS 1
```

will produce the dialogue below at the terminal. XPLANT will process the file as before but if any of the planted parameters VOL, DSN or TRACKS or the $JOBCARD parameters do not have the required type of value then XPLANT will prompt the user to enter a value of the correct type. In this example the value of LINES is not NUMERIC and VOL is not ALPHANUMERIC so XPLANT complains.

```
PUN  011 DEFINED
Enter keywords and values, terminated by null :-
lines a
vol rhel0*
```

```
dsn user.xxdata
jobname copy

Jobname "USCOPY" submitting to OS batch system, priority 12.
Value "A" for parameter "LINES" in file "TEST JOB A1 1" is not of
type "NUMBER"
Give value of "LINES"
1
Value "RHELO*" for parameter "VOL" in file "TEST JOB A1 9" is not
of type "VOLSER"
Give value of "VOL"
rhel01
PUN FILE 8570  TO  FEM      COPY 001   NOHOLD
PUN  011 DETACHED
R;
```

When the correct type of values have been given, XPLANT continues as before.

This example introduced the following ideas:-

(a) The **$JOBCARD** macro has a number of associated parameters. If any of these parameters are set prior to the macro call, either by setting them earlier in the file or on the command line, then these values are used. Otherwise the default values are used. The defaults are generally to put nothing on the jobcard and thus take the system default. These parameters are PH, ACCT, ID, PROGNAME, PRI, JOBNAME, TIME, LINES, CARDS, FORMS, COPIES, MSGLEVEL, LINECNT, NEEDS, COND and USERTEXT. Most of the names are self-explanatory but see Reference 1 or the CMS Help file XPLANT **$JOBFILE** for a fuller explanation.

(b) Macros may have an argument list. The call to the **$P** macro on line 10 of the CMS file has three arguments. The first, TRACKS, is the name of the parameter to be planted. The second, 5, is the default value which will be used if TRACKS is not set. The third, NUM, is a check-type (see point (d)). In the first bracket on line 9, the check-type is referred to by its name *CHECK because it is the third positional argument and the second is not given. An alternative would be to give the special value % for the second. This allows subsequent arguments to be given positionally without setting the second. The second bracket on this line is a call to **$P** with a default value given so the check type can be given as the third argument. For the names and positions of macro arguments see Reference 1 or the individual CMS Help files for each macro (eg for **$JOBCARD** type HELP XPLANT **$JOBCARD**)

(c) The XPLANT command has a PROMPT option which causes the user to be prompted for a list of parameter value pairs.

(d) The **$P** macro allows the checking of a parameter's value before planting it into the output file. If the value is not of the specified type, then the user is prompted for a value of the correct type and this value is subsequently used.

1.3 Example 3 - Multiple Versions of a Program

This example shows how to store a second version of a program in the same physical file.

The file listed below, contains a short FORTRAN77 subroutine.

```
      SUBROUTINE VNORM(A,N)
*     NORMALIZE VECTOR A , LENGTH N
      DIMENSION A(*)
*     MODULUS OF A
      AMODUL=0.0
      DO 10 I=1,N
10    AMODUL=A(I)**2+AMODUL
*     SCALE A
      FACT=1.0/SQRT(AMODUL)
      DO 20 I=1,N
20    A(I)=A(I)*FACT
      END
```

In ELECTRIC to create another version of this routine a separate edit file was created containing the edits to the original file. A short edit file for the above file is shown below.

```
FL=PYMAINDR.VNORM.ED  NENT=   7,NBLK=  1
 1 $G LN=   0.  1,DG=ONLY( 1) : ORIGINAL VERSION.
 1 $G LN=   0.  2,DP=ONLY( 2) : DOUBLE PRECISION VERSION.
 2 $I LN=   2.  1 :*      USE DOUBLE PRECISION INTERNALLY .
 2 $I LN=   3.  1 :       REAL*8 AMODUL,FACT
 2 $E LN=   5.  1,C1= 14,C2= 16 :0.D0
 2 $D LN=   9.  1,L2=   9
 2 $I LN=   9.  2 :       FACT=1.0D0/DSQRT(AMODUL)
EOP
```

When this file is used by ELECTRIC, the original version will be taken by default. To have the edits applied the edit group DP had to be specified.

With XPLANT, to create another version of this routine which is in the file VNORM XPLANT the XPLANT XEDIT macros are used when editing. The console listing for the editing session is shown below with a listing of the final file.

```
x vnorm xplant
EDITING FILE: VNORM XPLANT A1
XEDIT:
xplant double                                   .....set the xplant label
Warning: file made RECFM V
2
*   NORMALIZE VECTOR A , LENGTH N
xi                                               ......insert a line
INPUT MODE:
*     use double precision internally .
XEDIT:
1xi
INPUT MODE:
      real*8 amodul,fact
XEDIT:
2xchange/0.0/0.0d0/
      AMODUL={DOUBLE:0.0D0|:0.0}                .....change a string
```

```
/FACT/xrep          fact=1.0d0/dsqrt(amodul)  ......replace a line
file
R;
t vnorm xplant

      SUBROUTINE VNORM(A,N)
*      NORMALIZE VECTOR A , LENGTH N
{DOUBLE:*       USE DOUBLE PRECISION INTERNALLY .}
      DIMENSION A(*)
{DOUBLE:        REAL*8 AMODUL,FACT}
*     MODULUS OF A
      AMODUL={DOUBLE:0.0D0|:0.0}
      DO 10 I=1,N
10    AMODUL=A(I)**2+AMODUL
*     SCALE A
{DOUBLE:          FACT=1.0D0/DSQRT(AMODUL)|
:      FACT=1.0/SQRT(AMODUL)}
      DO 20 I=1,N
20    A(I)=A(I)*FACT
      END

R;
```

The command

```
XPLANT VNORM XPLANT A VNORM FORTRAN A
```

will copy the original version of this routine to the CMS file VNORM FORTRAN.

The command

```
XPLANT VNORM XPLANT A VNORM FORTRAN = (DOUBLE REPLACE
```

would overwrite VNORM FORTRAN with the alternative version of the routine.

1.4 Example 4 - Many Jobs in One File

In the previous examples the brackets have been simple macro calls (eg $P) which were unconditionally obeyed. XPLANT also allows conditional editing by the labelling of brackets and the combination of several macro calls in one bracket, only one of which may be executed.

XPLANT will treat a text string before a macro call (ie before a $) as a number of labels, eg -

```
{label1 label2 label3 $TYPE 'string1' }
```

This bracket has three labels : label1, label2, label3. $TYPE is a macro which outputs 'string1' to the terminal. XPLANT tests the values of the labels from left to right. If any of them have a value then $TYPE will be executed.

These labels are of two types.

(a) Labels with definite values. These are the parameters introduced earlier (eg VOL,DSN,TRACKS in Example 1).

(b) Labels used as 'switches' These can either be 'set' when their value is 'ON' or 'unset' when they have no value. These labels may be set ON by including them in the list of options (like SUBMIT and PROMPT) after the first left parenthesis on the command line.

The bracket may also be subdivided into sub-brackets or 'units'. These are separated by vertical bar characters (|). eg:

```
{L1 $TYPE 'string1'| L2 $TYPE 'string2'| L3 $TYPE 'string3'}
```

XPLANT scans the bracket from left to right and whenever it finds a unit that is active (ie one that has either a label with a value or no label at all), then it executes that unit and ignores the rest. So if L1 had a value then string1 would be $TYPEd. If L1 had no value and L2 did, then string2 would be typed etc.

The following example uses both of these features and another three macros.

(a) $SET — This macro sets a parameter to a given value.

```
{$SET A 1 $ A is set to 1 }
```

would give parameter A the value '1'. Note that the argument list for $SET is terminated by $. Any text following this second $ will be copied to the output stream with any brackets resolved.

(b) $GROUP — This macro will set a number of labels 'ON'. If the macro is made conditional, then setting one label on the command line would allow several more labels to be set inside the file.

```
{ G1 $GROUP L1 L2 L3 L4 L5 }
```

If G1 is set then L1, L2, L3, L4, L5 will be set also.

(c) $I — This macro does nothing but allows the temporary setting of parameters for other macro calls. It also has a special form of 'colon' (:) which is equivalent to I - ie output any text following.

```
{: A text string for output }
```

would copy the text string following the colon to the output stream with any imbedded brackets resolved.

The ELECTRIC file listed below, when used with its edit file, can create a job to copy either a dataset on tape or a catalogued disk dataset to a dataset on tape or disk by specifying one of four options TT, TD, DD or DT. In addition, the options SI or SO indicate that the input/output dataset is on a demountable disk.

```
FL=PYMAINDR.TESTJOB2  NENT=  12,NBLK=  1
  1://MYCOPYAB JOB (,,,),'GENERAL COPY JOB'
  2:/*SETUP INVOL,R,DEN6250
  3:/*SETUP DSN=INDSN
  4:/*SETUP OUTVOL,W,DEN6250
  5://*
  6://* A JOB TO COPY A DATASET FROM SETUP DISK TO SETUP DISK
  7://*
  8://STEP1 EXEC GENER
  9://IN  DD   DSN=INDSN,VOL=SER=INVOL,UNIT=DEN6250,
  10://          LABEL=(1,SL,,IN),DISP=OLD
  11://OUT  DD   DSN=OUTDSN,VOL=REF=OUTVOL,UNIT=DEN6250,
  12://          LABEL=(1,SL,,OUT),SPACE=(TRK,(5,5)),DISP=(NEW,CATLG)
 EOP

  FL=PYMAINDR.TESTJOB2.ED  NENT=  61,NBLK=  2
  1 $G LN=   0.  1,SO=ONLY( 9) :SETUP OUTPUT DISK
  1 $G LN=   0.  2,TD=ONLY( 1, 2, 5, 6,12) :      TAPE TO DISK
  1 $G LN=   0.  3,DD=ONLY( 1, 3, 5, 6, 7,11,12) :  DISK TO DISK
  1 $G LN=   0.  4,DT=ONLY( 1, 3, 4, 6, 7, 9,11,13) : DISK TO TAPE
  1 $G LN=   0.  5,TT=ONLY( 1, 2, 4, 6, 9) :      TAPE TO TAPE
  1 $G LN=   0.  6,SO= NOT( 5) : SETUP OUTPUT DISK
  1 $G LN=   0.  7,SI= NOT( 6) :  SETUP INPUT DISK
  1 $X LN=   0.  8 : OPTION TT -- COPY FROM TAPE TO TAPE
  1 $X LN=   0.  9 :        TD -- COPY FROM TAPE TO DISK
  1 $X LN=   0. 10 :        DD -- COPY FROM DISK TO DISK
  1 $X LN=   0. 11 :        DT -- COPY FROM DISK TO TAPE
  1 $X LN=   0. 12 :        SI -- INPUT DISK IS SETUP
  1 $X LN=   0. 13 :        SO -- OUTPUT DISK IS SETUP
  1 $X LN=   0. 14 :  LB=1   GENERAL EDITS
  2 $X LN=   0. 15 :  LB=2   TAPE INPUT
  3 $X LN=   0. 16 :  LB=3   DELETE INPUT SETUP
  4 $X LN=   0. 17 :  LB=4   TAPE OUTPUT
  5 $X LN=   0. 18 :  LB=5   DELETE OUTPUT SETUP
  6 $X LN=   0. 19 :  LB=6   DELETE SETUP DSN
  7 $X LN=   0. 20 :  LB=7   DISK INPUT
  9 $X LN=   0. 21 :  LB=9   TAPE OUTPUT
 11 $X LN=   0. 22 :  LN=11  DISK INPUT DELETE VOL=
 12 $X LN=   0. 23 :  LB=12  DISK OUTPUT
  2 $E LN=   1.  1,C1=  9,C2=  9 :T
  7 $E LN=   1.  2,C1=  9,C2=  9 :D
  4 $E LN=   1.  3,C1= 10,C2= 10 :T
 12 $E LN=   1.  4,C1= 10,C2= 10 :D
  3 $D LN=   2.  1,L2=   2
  2 $P LN=   2.  2,C1=  9,C2= 13,CH=NO,DF=NO,NM=INVOL
  1 $P LN=   2.  3,C1= 17,C2= 23,CH=NO,DF=YS,NM=INUNIT
  6 $D LN=   3.  1,L2=   3
  1 $P LN=   3.  2,C1= 13,C2= 17,CH=DS,DF=NO,NM=INDSN
  5 $D LN=   4.  1,L2=   4
  1 $P LN=   4.  2,C1=  9,C2= 14,CH=NO,DF=NO,NM=OUTVOL
  1 $P LN=   4.  3,C1= 18,C2= 25,CH=NO,DF=YS,NM=OUTUNIT
  2 $E LN=   6.  1,C1= 34,C2= 43 :TAPE
  6 $E LN=   6.  2,C1= 34,C2= 38 :
  5 $E LN=   6.  3,C1= 48,C2= 52 :
  4 $E LN=   6.  4,C1= 48,C2= 57 :TAPE
  1 $P LN=   9.  1,C1= 16,C2= 20,CH=DS,DF=NO,NM=INDSN
 11 $E LN=   9.  2,C1= 22,C2= 35 :
  2 $P LN=   9.  3,C1= 30,C2= 34,CH=NO,DF=NO,NM=INVOL
 11 $E LN=   9.  4,C1= 36,C2= 48 :
  2 $P LN=   9.  5,C1= 41,C2= 47,CH=NO,DF=YS,NM=INUNIT
  7 $E LN=  10.  1,C1= 12,C2= 28 :
  2 $P LN=  10.  2,C1= 19,C2= 19,CH=NO,DF=YS,NM=INLABEL
  1 $P LN=  10.  3,C1= 34,C2= 36,CH=NO,DF=YS,NM=INDISP
  1 $P LN=  11.  1,C1= 16,C2= 21,CH=DS,DF=NO,NM=OUTDSN
  9 $E LN=  11.  2,C1= 27,C2= 29 .SER
  1 $P LN=  11.  3,C1= 31,C2= 36,CH=NO,DF=NO,NM=OUTVOL
  5 $E LN=  11.  4,C1= 38,C2= 50 :
  9 $P LN=  11.  5,C1= 43,C2= 49,CH=NO,DF=YS,NM=OUTUNIT
 12 $E LN=  12.  1,C1= 12,C2= 29 :
  4 $P LN=  12.  2,C1= 19,C2= 19,CH=NO,DF=NO,NM=OUTLABEL
  4 $E LN=  12.  3,C1= 30,C2= 47 :
 12 $P LN=  12.  4,C1= 37,C2= 39,CH=NO,DF=YS,NM=ALLOC
 12 $P LN=  12.  5,C1= 42,C2= 42,CH=NO,DF=YS,NM=PRIMALLO
 12 $P LN=  12.  6,C1= 44,C2= 44,CH=NO,DF=YS,NM=SECALLO
  1 $P LN=  12.  7,C1= 54,C2= 56,CH=NO,DF=YS,NM=OUTDISP
  4 $E LN=  12.  8,C1= 58,C2= 62 :KEEP
  1 $P LN=  12.  9,C1= 58,C2= 62,CH=NO,DF=YS,NM=DISPAFT
EOP
```

This is achieved by labelling the edits in the edit file in distinct numbered sets and then grouping these sets to provide the edits required for an option.

For example, ELECTRIC command

```
PARM INVOL=GCG001,OUTVOL=938571,INDSN=TAPE.DATA1,OUTDSN=TAPE.DATA2
PARM INLABEL=3,OUTLABEL=2,INUNIT=DEN1600
COPY TESTJOB2(TT),TESTOUT
```

would create the following ELECTRIC file.

```
//MYCOPYTT JOB (,,,),'GENERAL COPY JOB'
/*SETUP GCG001,R,DEN1600
/*SETUP 938571,W,DEN6250
//*
//* A JOB TO COPY A DATASET FROM TAPE TO TAPE
//*
//STEP1 EXEC GENER
//IN  DD   DSN=TAPE.DATA1,VOL=SER=GCG001,UNIT=DEN1600,
//         LABEL=(1,SL,,IN),DISP=OLD
//OUT  DD  DSN=TAPE.DATA2,VOL=SER=938571,UNIT=DEN6250,
//         LABEL=(1,SL,,OUT),DISP=(NEW,KEEP)
```

The edit group TT selects the appropriate edits for a job to copy a dataset from tape to tape.
Similarly, the command

```
PARM OUTVOL=RHEL04,INDSN=USER.DISK03
PARM OUTDSN=USER.DISK04
COPY TESTJOB2(DD,SI),TESTOUT2
```

would select the edits to create the following file.

```
//MYCOPYDD JOB (,,,),'GENERAL COPY JOB'
/*SETUP DSN=USER.DISK03
//*
//* A JOB TO COPY A DATASET FROM SETUP DISK TO  DISK
//*
//STEP1 EXEC GENER
//IN  DD   DSN=USER.DISK03,
//         DISP=OLD
//OUT  DD  DSN=USER.DISK04,VOL=REF=RHEL04,
//         SPACE=(TRK,(5,5)),DISP=(NEW,CATLG)
```

An XPLANT file to perform a similar function would look as follows.

```
{TAPETAPE $GROUP TI TO}                                            ...(b)
{TAPEDISK $GROUP TI DO}
{DISKDISK $GROUP DI DO}
{DISKTAPE $GROUP DI TO}
{$SET JOBNAME COPY{TI:T|DI:D}{TO:T|DO:D}}                           ...(c)
{$JOBCARD}
//*
//* COPY A D/S FROM {SI:SETUP }{TI:TAPE|DI:DISK} TO {SO
:SETUP }{TO:TAPE|DO:DISK}                                          ...(d)
//*
{TI :/*SETUP {$P INVOL % VOLSER},R,{INUNIT|:DEN6250}|              ...(e)
 SI :/*SETUP DSN={$P INDSN % OSDSN}}
{TO SO :/*SETUP {$P OUTVOL *CHECK=VOLSER},W,{OUTUNIT|:DEN6250}}
//STEP1 EXEC GENER
//IN DD DSN={SI:{INDSN}|$P INDSN % OSDSN},                         ...(f)
{TI :// VOL=SER={INVOL},UNIT={INUNIT|:DEN6250},LABEL=({INLABEL...(g)
|:1},SL,,IN),}
// DISP=OLD
//OUT DD DSN={$P OUTDSN % OSDSN},VOL={TO SO:SER|:REF}={OUTVOL},
//      {TO :LABEL=({$P OUTLABEL % N},SL,,OUT)|
          DO :SPACE=({ALLOC|:TRK},({PRIMALLO|:5},{SECALLO|:5}))},
//    {TO SO:UNIT={OUTUNIT|:DEN6250},{DISP=({DISP                  ...(h)
|:NEW},{DISPAFT|TO:KEEP|:CATLG})
```

If a second file is named on the XPLANT command line, XPLANT performs a copy to a CMS file similar to ELECTRIC's COPY with edits.

The CMS command

```
XPLANT COPY JOB A TESTOUT (TAPETAPE PROMPT
```

would prompt the user to enter the parameter values.

```
Enter keywords and values, terminated by null :-
INVOL GCG001
OUTVOL 938571
INDSN TAPE.DATA1
OUTDSN TAPE.DATA2
INLABEL 1
OUTLABEL 1
INUNIT DEN1600

OS jobname is "PYCOPYTT"
R;
```

would produce the following output in the CMS file TESTOUT JOB A (Filetype and filemode default to those of the input file.)

```
//PYCOPYTT JOB (4300,PY,,,,,,,),
// 'VM/JCG/PH-33'
//* Command line:XPLANT COPY JOB A TESTOUT ( TAPETAPE PROMPT
//* $JOBCARD called from "COPY     JOB     A1 7"
//* Submitted from RAL CMS userid "JCG" on 11/12/82 17:24:08
//*
//* COPY A D/S FROM TAPE TO TAPE
```

```
//*
/*SETUP GCG001,R,DEN1600
/*SETUP 938571,W,DEN6250
//STEP1 EXEC GENER
//IN DD DSN=TAPE.DATA1,
//   VOL=SER=GCG001,UNIT=DEN1600,LABEL=(1,SL,,IN),
//      DISP=OLD
//OUT DD DSN=TAPE.DATA2,VOL=SER=938571,
//        LABEL=(1,SL,,OUT),
//     UNIT=DEN6250,DISP=(NEW,KEEP)
```

Note the following points:

(a) The options are not restricted to two characters so they can be given more self-explanatory names, like TAPEDISK instead of TD.

(b) Depending on the option selected, two other labels are set. These are then used as labels in several other edits. This was done for brevity. The label TI could be replaced everywhere by the two labels TAPETAPE and TAPEDISK but the file would then become more cumbersome.

(c) The name of the job may be given by setting the parameter JOBNAME or as the second positional parameter in a call to the $JOBCARD macro.

(d) Line 6. The labels TO and DO are mutually exclusive so they can be used in the same bracket.

(e) /*SETUP etc will be output if label TI is set. If INUNIT has a value it will be output.

(f) If SI is set then INDSN was set three lines previously so $P need not be used, otherwise call $P to check for a value of DSNIN.

(g) If the label INLABEL has a value, it will be output, otherwise the text string '1' will be output. This is an alternative method of specifying a default value for a parameter without type checking. This method is used several times (eg ALLOC, PRIMALLO, SECALLO on Line 21).

(h) The first bracket shows the use of several labels in a bracket.

Compare the length of the XPLANT file on page 11 with the combined length of the ELECTRIC file and edit file listed starting on page 8.

2. XPLANT

This section describes in more detail the structure of XPLANT edits. It defines the terms bracket, label and unit and shows the use of the XPLANT command. The user is then given a brief description of a few useful XPLANT macros and introduced to the set of XEDIT macros specially written to help the creation of XPLANT brackets.

2.1 Brackets

A 'bracket' is started by a special character and terminated by another. At the start of the input file these characters are the left 'curly' bracket ({) hex 8B and the right 'curly' bracket (}) hex 9B but these, and other special characters, may be changed at any time during the processing of the file.

The term 'bracket' is used to refer to these special characters and all characters between them. Within a bracket, the characters dollar (**$**),vertical bar (|) and colon(:) are also recognised as special.

2.1.1 Bracket Processing

A bracket may be subdivided into any number of 'units'. These units are separated by vertical bars.

bracket	{ unit1 | unit2 | ..... unitn }

Only one, if any, of these units will be processed.

The bracket is scanned from left to right until an executable (or 'active') unit is found (See Section 2.2). That unit is then executed and the rest of the bracket is skipped. If no active units are found then the bracket produces no output. Each bracket may contain any number of nested brackets, the inner brackets being processed first. Line feeds within a bracket are ignored unless they are part of a text string, in which case they will be output.

2.1.2 Unit Processing

Each unit within a bracket can take one of the following forms:

```
   unit         labels
                labels $opcode parameters
                       $opcode parameters
                       $opcode parameters $text
                labels $opcode parameters $text

                labels  :text
                        :text
```

where:

labels — Each label is a string of characters up to 255 long (special characters in quotes). (see the following section.)

opcode — This is the name of the XPLANT function or macro to be executed. User defined macros may also be used here. See Appendix A for a one-line description of each system provided function, and Section 4 and Reference 1 for more details on macros. Information on all macros and functions is also available in CMS by typing -- Help XPLANT $opcode

The colon (:) form is simply a special opcode equivalent to "I" without any parameter settings.

parameters — These are the parameter settings passed to the macro. The settings may be positional and/or keyword but special characters (including space in a value) must be enclosed in single quotes. Positional parameters may be skipped (and not set) by providing the special value of "%".

text — This text is introduced by the "$" which also terminates the parameters. It is processed as normal (any internal brackets being resolved on the way) and copied to the output stream.

2.2 Labels

Labels are names which can occur at the beginning of each "unit" in a bracket processed by XPLANT (see Section 2.1) and they control the operation of that unit within the bracket. If one or more of those names have a value then the labels are considered "on" and that unit will be processed and all subsequent units within that bracket will be skipped. Note that only one unit out of the bracket is executed at most.

2.2.1 Examples

The macros used have the following effects;

> $TYPE - outputs text to the terminal.
> $SET A B - sets parameter A to the value 'B'

(a) {X} - returns the value of X

(b) {X Y} - returns the value of X if it has one. If not it returns the value of Y if it has one. If neither X nor Y have values then nothing is returned.

(c) {$SET A 123} - gives parameter A the value '123' but returns nothing.

(d) {X$SET A {X}} - If X has a value, then set A to the value of X. Here X is used both as a label and as a parameter value.

(e) {$SET A 123 $ A = {A}} - unconditionally set A to 123 then output the text string 'A = 123'. nb {A} will return '123'.

(f) {A:A has a value|:A has no value} - If A has been set, either as in example (c) or on the command line, then the first unit is active and the bracket will return the text string 'A has a value'. If A has not been set, then the first unit is skipped, the second is processed and the text string 'A has no value' is output.

2.3 The XPLANT Command

The XPLANT command has the format

```
┌────────┬──────────────────────────────────────────────────────────────┐
│        │                                                              │
│ XPlant │ fn ft fm  ofn oft ofm  (booleans (key-value pairs            │
│        │                                                              │
└────────┴──────────────────────────────────────────────────────────────┘
```

Where

fn,ft,fm — name type and mode of the file to be planted. Default of XPLANT,XPLANT,*.

ofn,oft,ofm — name type and mode of the output file. Default values are the name, type and mode of the input file unless the input filetype is XPLANT in which case the output filetype used is XPLANTED.

Note that all the filenames are optional, and that omitted components of the output file default to values related to the corresponding component of the input file.

XPLANT allows two types of option.

(a) Simple switches (or booleans). These are options which are set to a non-null value if specified by the user and unset if not.

(b) Options with values. These option names are set to the given value for the duration of the XPLANT run or until reset or unset.

```
XPLANT MY JOB A ( OPTIONA ( OPTIONB valueb ....
```

The OPTIONA form is useful for setting a user's labels 'on'. They can then be used to control the processing of brackets. There are also a number of system boolean options which have no values but are just executed if given.

File Causes output to be written into the file. FILE is implied if neither TERM nor SUBMIT nor PRINT are specified, or if part of the output filename is supplied.

Terminal Causes output to appear on the terminal.

Print Causes the output to be printed on the virtual printer.

PRT Is an alternative to PRINT.

Submit Causes the output to be submitted as a job to the OS system defined by the CMS VROUTE EXEC. This implies VALUES (qv).

TRACE Causes trace output of XPLANT operations to be written to the console as planting proceeds. Useful for debugging.

DUMP Causes the values of the variables currently set to be displayed if an error occurs.

NOUse Prevents an error if there are any $SETed variables left at the end of the XPLANT run that have not been used. The default is to check them and end with a non-zero return code (and to purge the submitted file). The error message specifies which variables were set and their values.

VALues Causes a list of the names and values of the user variables that have been used during the XPLANT run to be generated at the end of the output file in the form of JCL comments. This is implied by the SUBMIT boolean.

NOValues Suppresses the JCL comments that would normally be created by SUBMIT or VALUES.

Mixed Causes information read from the terminal to be left in mixed case. This is the default for files with filetype MEMO, GEROFF, SCRIPT or LAYOUT. For other filetypes, information read from the console is uppercased.

PRompt Causes a request for keywords and values to be entered from the console as XPLANT starts up. The first token is the keyword to be set, the remainder of the line (after the first space) is the value to which it is set Quotes (') are not allowed in these lines.

REPlace Unless REPLACE or APPEND is specified, the output file must not already exist. If REPLACE is specified, and the output file already exists, it is erased and a new one created.

APPend — Unless REPLACE or APPEND is specified, the output file must not already exist. If APPEND is specified, and the output file already exists, the new output is appended to the end of the existing file. If the output file does not already exist, then it is created. RECFM or LRECL cannot be specified with APPEND.

The OPTIONB form is used to set parameters to a value for the duration of the XPLANT run. Values so set can be used as labels or processed by $P or any of the other macros. There are also a number specially recognised system options with values. These are shown below.

RECFM F|V — The value of this should be F or V, and is the recfm of the newly created output file. The default is F.

LRECL n — Record length of the output file (in the range 1 to 256). The default for LRECL is 80. N.B. LRECL is ignored if the output has recfm V.

SETFILE fn — The name of a file containing variables to be set (like PROMPT above, but taken from a file). The filetype of the file is always XPLANT and the filemode *. See the $SETFILE function for the format to be used inside the file.

MEMBER membername — If the input file specified is a MACLIB, this membername if the name of the member of the maclib that is actually XPLANTed. If it is ommitted, the whole file is processed.

ROUte pri <sec> — This option is used with the "SUBMIT" option to allow you to temporarily override the default routing which has been defined earlier by calling the VROUTE EXEC. The routing defined here or the default set with VROUTE will effect all jobs within the file unless overridden by /*ROUTE cards within the job. Note that if the secondary address is given, this parameter cannot be given on the XPLANT command line because only one value is permitted per parameter. To give both addresses, use the "PROMPT" option or supply the value of "ROUTE" in a file and use the "SETFILE" option. See also the SECROUTE parameter below. Note that if the VROUTE EXEC is used to direct job submission to another batch system, then this parameter will be ignored.

SECROUTE sec — This option is used in a similar way to "ROUTE" but only overrides the secondary routing address.

TOCMSID uid — This option is exactly equivalent to specifying "ROUTE RLVM370 uid" which will cause the default routing of output from all jobs sent in the spool file, to be sent back to the virtual reader of the specified CMS user (provided the individual jobs do not contain overriding /*ROUTE cards). N.B. This parameter can only be used when job submission is to the RAL OS/MVT system i.e. the value of SUBROUTE (see below) is (or defaults to) RLVM370.

SUBRoute n
: This option temporarily overrides the routing of the job input. If "n" is a valid NJE-linkid known to the RAL VNET system, (e.g. CERN whose linkid is GEN), then this job will be submitted down that link to run at the remote site. Note that the special value RLVM370 for "n" represents the RAL OS/MVT system. Only when the value of SUBROUTE is (or defaults to) RLVM370 can the output routing options (ROUTE, TOCMSID see above) be used.

MACRO memname
: If this parameter is set to the member name of a macro in one of the maclibs available to XPLANT, then this macro will be executed using only the arguments set on the command line. This can be useful for testing macros in maclibs. See "HELP XPLANT MACLIB" for details about how to make maclibs available to XPLANT.

2.3.1 Examples

(a) Submitting a job to the OS batch system:

```
XPLANT COMPILE JOB ( SUBMIT
```

This will process the CMS file "COMPILE JOB" (including any other files which may be $ADDed in) and punch the output to the OS batch machine, tagged correctly to route the output according to your VROUTE settings. If the VROUTE SUBMIT setting is not the local RAL Batch system then the settings of the ROUTE, SECROUTE and TOCMSID options are not obeyed and the job must contain a /*ROUTE card (or equivalent).

(b) Copying to a new CMS file:

```
XPLANT ANALYSE XPLANT * TEMP FORTRAN A
```

The CMS file "ANALYSE XPLANT *" is processed and the output is written to a new CMS file "TEMP FORTRAN A". The default output format for CMS files is recfm F, lrecl 80 so that it is convenient for feeding into the CMS compilers which demand fixed length record input. An error would be issue for this command if the output file already existed.

(c) Outputting on the terminal and setting some labels and parameters:

```
XPLANT TEST ( TERMINAL NOLKED ( COMP H
```

This performs a copy from file "TEST XPLANT *" to the terminal - this can be a useful way of testing the effect of plants and edits. After the first left bracket are two labels; "terminal" is the system label which causes the output to be directed to the terminal. "nolked" is a private label which could (for example) prevent the execution of the link-edit step in a job contained in the file. After the second left bracket is one key-value pair which sets "COMP" to have the value "H". Thus within the files processed by XPLANT in this example, "{COMP}" will be replaced by "H" unless the parameter "COMP" is unset or reset.

2.4 $P

This macro returns the value of a variable which has been previously set, it can take a default or can prompt for a value. Value checking can be performed. This is the equivalent of ELECTRIC's **$P** plant command.

The arguments for the **$P** function are :-

```
{$P *PARM=name *DEFPARM=name *CHECK=type *MESS='prompt'}
```

Parameters

*PARM This is the name of the variable whose value is to be determined or checked. The variable may have been given a value on the command line or internally prior to execution of this macro. If it is not set at this point, the default is taken if one is set, otherwise an error prompt appears for the value to be given

*DEFPARM This is the default value to be used if the parameter specified for *PARM has no value. It is not checked against the check-type and may be null (for a null default) or omitted to force a value to be given because no default is allowed. If given as "%", it is considered unset.

*CHECK This is the check type for the value of the variable named in *PARM. See Section 2.7 for a list of check types available. If this check type is omitted, no checking is performed. If the resulting value does not correspond to the check type, an error message is sent to the terminal followed by a prompt for the value to be given again. Note that if the label NOPROMPT is set on, then this prompt will not occur and the XPLANT program will exit with condition code 1.

*MESS This is the prompt to be used for the value if the variable named in *PARM has no value. If it is absent, no prompt is given. If it is set to "YES", then the default prompt is used. N.B. a null reply to this prompt will cause the default to be taken if a default was specified.

Operation of the **$P** macro:

(a) If the variable whose name is specified as *PARM is already set, its value is taken and is checked against the check-type if this was specified. If no check-type was given, no checking is performed and the value is returned. If the value does not match the check-type, an error message appears on the terminal followed by a prompt to give the value again. Only a value matching the check-type will be finally accepted.

(b) If the variable whose name is specified as "*PARM" is not set, the default value is taken if it exists. This default value is not checked against any check-type even if this was given. If the default is omitted or specified as "%", no default is allowed. A terminal error message appears followed by a prompt to specify the value. This value must match the check-type if the one was specified.

(c) If the label "*NOPROMPT" is set (e.g. on the command line) the macro will not prompt if the value is of incorrect type or if no default is allowed, but will end the program with condition code 1. This can be of use if the program is run in a batch system.

(d) nb If the default value is taken, the parameter name is NOT set (with $SET - see HELP XPLANT $SET) so that different defaults may be taken for different calls to the $P macro. However if the value does not fit the check-type, or if no default is permitted, then the parameter IS set (with $SET) so that later calls to macro get the new (possibly corrected) value and will accept it without further prompting.

2.4.1 Examples

Consider the following lines of JCL.

```
//FT01F001 DD VOL=SER={$P VOL USDSK1 VOLSER},DSN=USER.FRED
//FT02F001 DD VOL=SER={$P VOL RHEL02 VOLSER}
//FT03F001 DD VOL=SER={VOL},DSN=USER.JOE
```

If the variable "VOL" was set (eg on the command line), then this will be accepted and used by the two $P calls provided the value was of type "VOLSER". If not, a terminal prompt would request the value to be given again and this new value would override the old value and be accepted by the call on the second DD card. nb if "VOL" was NOT set before XPLANT processes these lines, both defaults for the two calls will be taken and variable "VOL" will NOT be set (and so a null would be returned in the third DDcard).

Values may be passed to the macro as positional or keyword parameters, so the bracket on the first line could have been written as

```
      {$P VOL USDSK1 *CHECK=VOLSER}
or    {$P VOL *CHECK=VOLSER *DEFPARM=USDSK1}
or    {$P *CHECK=VOLSER *DEFPARM=USDSK1 *PARM=VOL}
```

```
// DSN={$P DSN % OSDSN}
```

The "DSN" call to $P has no default (because "%" means not set), so if "DSN" is not set before, an error prompt will be produced and the value accepted (which must be of type "OSDSN"), would be $SETed so that later calls to the macro would use it (provided they also required values of type "OSDSN").

```
// DSN={$P DSN % OSDSN 'Give name of your dataset'}
```

This call behaves exactly as above, but will prompt you for the dataset unless the variable has been previously set.

```
// DSN={$P DSN}
```

This call will return the value of DSN if it is set but prompt you with an error message if not. No checking is performed on the value at all.

2.5 Special Xplant Xedit Macros

To help create files for processing by XPLANT, two sets of XEDIT macros have been written.

One set of macros closely mirror existing XEDIT subcommands (like Change), except that they perform the operation conditionally. These editing commands all begin with "X" followed by the name of the normal XEDIT command for that function. So XCHANGE is the conditional form of CHANGE and XREPLACE is the conditional form of REPLACE etc. These commands have identical syntax to the corresponding XEDIT commands but instead of immediately performing the edits, they insert a bracket in the file which will perform the required edit when the file is processed by XPLANT. The edit will be conditional on a label. This label is the one currently set by the XPLANT xedit macro (see below).

The second group of subcommands provide the ability to insert plants ($P commands),supply files ($ADD commands) and group definitions (like ELECTRIC's groups) into a file. These do not mirror existing commands and have their own separate help files. The names are listed below and in the XEDIT menu.

Before using any of these special commands in an editing session, it is essential to call the XPLANT sub-command. This sets up synonyms and provides an opportunity to set or change the editing label.

The XPLANT xedit command has the form

```
┌───────────┬──────────────────────────────────────────────────────┐
│           │                                                      │
│  XPLANT   │  <label <left-bracket <right-bracket>>>              │
│           │                {              {                      │
│           │                                                      │
└───────────┴──────────────────────────────────────────────────────┘
```

where label is the current XPLANT label and left and right bracket are the XPLANT bracket delimiters as defined in Section 2.1. nb The conditional edit commands do NOT take account of existing edits in the file, so it is the user's responsibility to make sure the brackets remain properly nested. Given these restrictions, it is possible to use these edit commands to perform simple editing and even to edit existing edits (see examples below).

As a result of the conditional edits, lines will become longer and it may be necessary to invoke XEDIT with an increased WIDTH option to permit more edits to be inserted.

Note that only a subset of the XEDIT commands have conditional forms and these appear in the list below with their minimum abbreviation shown in capitals.

```
XAfter   XAPpend   XBefore  XCAppend  XChange   XCInsert  XCOpy
XDelete  XDUplica  XInput   XMOve     XReplace  XSElect   XUnder
```

Below is a list of other XEDIT subcommands which are useful for planting, adding and group definitions:

GRoup inserts a group definition at the top of the current file. Line pointer is repositioned after the insert.

PLant inserts one or more $P brackets. Existing strings in the file become the default strings of the plants.

PLAFter inserts one or more $P brackets. The plant brackets are inserted AFTER the specified strings in the file and do not replace them.

PLBEfore inserts one or more $P brackets. The plant brackets are inserted BEFORE the specified strings in the file and do not replace them.

ENCLOSE makes an existing block of text look like a conditional insert.

SUPPLY add a CMS file to the output stream.

2.5.1 Examples of conditional editing.

Here is the console from a short editing session to modify a short job for use by XPLANT.

```
t general job

/*PRIORITY 12
//MYJOB JOB (XXXX,ID),'JOB'
//*    A NULL JOB
//STEP1 EXEC NULL

R;
xedit general job a
xplant xt
Warning: Serialisation removed and file made RECFM V
1xc/12/10/
/*PRIORITY  {XT:10|:12}
1xc/my/us/
//{XT:US|:MY}JOB JOB (XXXX,ID),'JOB'
xc/xxxx/1234/
//{XT:US|:MY}JOB JOB ({XT:1234|:XXXX},ID),'JOB'
xc/id/us/
//{XT:US|:MY}JOB JOB ({XT:1234|:XXXX},{XT:US|:ID}),'JOB'
xb/job/xtape / 1 1 3
//{XT:US|:MY}JOB JOB ({XT:1234|:XXXX},{XT:US|:ID}),'{XT:XTAPE} JOB'
1xdel 1
{XT://*   RUN XTAPE.}
xi //*   run xtape.
xi /*setup  {vol}
1xc/null /XTAPE,TAPE={VOL},OPT=E2/
//STEP1 EXEC {XT:XTAPE,TAPE={VOL},OPT=E2|:NULL}
fil
R;
```

```
type

/*PRIORITY {XT:10|:12}
//{XT:US|:MY}JOB JOB ({XT:1234|:XXXX},{XT:US|:ID}),'{XT:XTAPE|:A} JOB
{XT://*    RUN XTAPE.}
{XT:/*SETUP  {VOL}}
//STEP1 EXEC {XT:XTAPE,TAPE={VOL},OPT=E2|:NULL}

R;
xplant general job (term
/*PRIORITY 12
//MYJOB JOB (XXXX,ID),'GENERAL JOB'
//*     A NULL JOB
//STEP1 EXEC NULL
R;
xplant general job (term xt (vol 123456
/*PRIORITY 10
//USJOB JOB (1234,US),'XTAPE JOB'
//*    RUN XTAPE.
/*SETUP  123456
//STEP1 EXEC XTAPE,TAPE=123456,OPT=E2
R;
```

2.6 $JOBCARD

Returns a /*PRIORITY and HASP // JOB card. All parameters can be planted and values are checked.

The arguments for the $JOBCARD function are :-

```
{$JOBCARD *PRI=pri *JBNM=jbnm *TIME=time *LINES=lines
          *CARDS=cards *FORMS=forms *COPIES=copies
          *MSGLEVEL=msglevel *LCNT=linecount *NEEDS=needs}

          Override with parameters:
          PH, ACCT, ID, PROGNAME,
          PRI, JOBNAME, TIME, LINES, CARDS, FORMS, COPIES,
          MSGLEVEL, LINECNT and NEEDS

          see also parameter "COND" below
```

2.6.1 Parameters

*PRI This is the the priority for the /*PRIORITY card.

*JBNM This parameter will form the last part of the OS jobname (the first part being the pigeon-hole parameter) This parameter may be overridden by setting "JOBNAME".

*TIME This is the HASP job time limit.

*LINES This is the HASP thou-line limit.

*CARDS This is the HASP card limit.

*FORMS This is the HASP forms parameter.

*COPIES This is the HASP copies parameter.

*MSGLEVEL This is the HASP message-level control including the brackets.

*LCNT This is the HASP linecount parameter which must be numeric. It may be overridden by setting parameter "LINECNT"

*NEEDS If this or parameter "NEEDS" is set (say on the command line), a "/*NEEDS" card will be inserted. The value is not checked, but the value of "NEEDS" overrides the default set with "*NEEDS" on the call to this macro.

COND If this parameter is set (say on the command line), then a "/*COND" card is inserted containing the value of COND.

2.6.2 User Dependent Parameters (PH, ACCT, ID, PROGNAME)

The defaults for the parameters pigeon-hole (which forms the first part of the jobname), the OS account and identifer and the programmer name are all obtained from a QSET variable called SYJBCARD. They may all be overridden by settings of the above names on the XPLANT command line. The QSET variable may be set up conveniently by calling the CMS EXEC JOBINFO from the PROFILE EXEC (See HELP CMS JOBINFO).

The macro is executed in three stages as follows.

(a) The /*PRIORITY and JOB cards are created using the overridden parameters (from the command line) and taking defaults from the the macro call. Nothing is inserted for parameters like CARDS if the value is not overridden and there is no default.

(b) JCL comment cards are also created giving the original XPLANT command line, the fileid and line number of the file from which the $JOBCARD macro was called and the submitting CMS userid, date and time. A message to the terminal also informs you what the final jobname is.

(c) Any parameter values which do not fit the correct check-types, will be faulted and a prompt will be issued for the parameter(s) to be given again. The new value will be $SETed to override the faulty value.

2.6.3 Examples

```
      {$JOBCARD 12 TEST 1-30 % 200 555}
```

with the following XPLANT command:

```
      XPLANT TEST JOB (SUBMIT (TIME 2 COPIES 3
```

would produce the following JCL.

```
      /*PRIORITY 12
      //LRTEST JOB (1234,ab,2,,200,555,3,,),
      // 'VM/DEMO/PH-xx'
      //* Command line:XPLANT TEST JOB (SUBMIT (TIME 2 COPIES 3
      //* $JOBCARD called from "TEST     XPLANT   A1 1"
      //* Submitted from RAL CMS userid "DEMO" on 01/19/83 14:24:39
```

The LINES parameter has been skipped in the macro call by giving the value as "%". The "TIME" and "COPIES" defaults have been overridden.

2.7 $CHECK

Checks that a string is of a valid format for a specific object.

The arguments for the $CHECK function are :-

```
{$CHECK *TXT=text *TYPE=check-type}
```

where

*TXT is the string to be tested.

*TYPE is the check-type required. This is a substring of the check-types described below, with the minimum length shown in capitals.

$CHECK returns YES or NO according to whether *TXT is of the type specified by *TYPE, or not.

The possible check-types are

Alpha (a string consisting of either case letters)

ALPHANum or AN (a string consisting of upper case letters and digits)

ANY this matches any string

Ddname (a valid OS/MVT DDname)

File (a valid CMS file-identifier (filename filetype {filemode}))

FLoat (a valid FORTRAN floating point number (with or without exp)

FM	(a valid CMS filemode)
FN	(a valid CMS filename)
FT	(a valid CMS filetype)
Integer	(a string consisting of digits, maybe with a sign)
Number	(a string consisting of digits)
Osdsn	(a valid OS/MVT dataset name, without a member name)
OSPds	(a valid OS/MVT dataset name, with or without a member name)
Volser	(a valid OS 360 volume serial number)

Example

```
{{$CHECK '{XXX}' OSP}$T 'OS PDS name is {XXX}'}
```

will only type the message if XXX is a valid OS dataset name, possibly with a member name/generation data group.

2.8 $SET and $UNSET

$SET

Sets a variable to a value.

```
┌─────────────────────────────┐
|                             |
|  {$SET *A=name *B=value}    |
|                             |
└─────────────────────────────┘
```

The name, the value of *A has its value set to the value of *B. This setting remains until the variable is reset, by $SET or $UNSET. $SET returns nothing.

For example, if at the top of the XPLANT file, {$SET X YYY} occurs, then from there on, {X} will become YYY.

$UNSET

Unsets a variable.

```
┌──────────────────────┐
|                      |
|  {$UNSET *A=name}    |
|                      |
└──────────────────────┘
```

The specified name has its value removed. The name must have been $SET, not just locally bound. Note that if a variable has been $SET more than once, the value does not return to its previous value; the variable does not have any SET value after $UNSET.

For example,

```
{$SET X YYY}
{X}
{$UNSET X}
{X}
```

The first {X} returns YYY, but the second returns nothing at all

2.9 $ADD

Includes all or part of a specified file.

The arguments for the $ADD ($A) function are

```
{$ADD *FN=fn *FT=ft *FM=fm *FROM=ln *FOR=n  *PLANT=<yes|no>}
```

where

*FN,*FT,*FM are the filename,filetype and filemode of the file to be included. If * is specified, then the file with the specified name first in the CMS search order is included. The default values are those of the file from which the call to $ADD was made.

*FROM is the line number of the first line to be included.

*FOR is the maximum number of lines to be included.

*PLANT should be YES or NO. The file included is searched for plants and these are performed, if and only if *PLANT is YES. *PLANT=YES is the initial value.

Usage Notes

A file can $ADD itself, recursively.

The output from $ADD (ie the contents of the file) can be used as input to other functions.

For more information and examples, see section 3.2.

2.10 $NOT

Performs a 'NOT' function on the specified argument.

The arguments for the $NOT function are

```
{$NOT *A=name}
```

$NOT returns "NO" if the name specified by *A has a value, otherwise it returns "YES".

For example,

```
{{$NOT FOO}$T 'foo was not set'}
```

will type the message if and only if the variable "FOO" does not have a value.

2.11 $EXIT

Terminates execution of XPLANT with a return command.

The arguments for the $EXIT function are

```
{$EXIT *RETCODE=number}
```

Xplant stops executing, with the specified return code.

$STOP is a synonym for $EXIT

3. REPLACEMENTS FOR ELECTRIC DELAYED EDITING COMMANDS.

This section lists the ELECTRIC editing commands which are relevant to delayed editing and suggests ways of reproducing their effect when using XPLANT. These should suffice for the user who persists in thinking in terms of 'ELECTRIC-type' editing but once he becomes familiar with XPLANT, the user will find simpler ways of achieving the same results and avoid the complex and often cumbersome edit files which often occurred with ELECTRIC.

To summarize this section;

(a) The user who wishes to create an alternative version of a file should use the XPLANT XEDIT macros.

First call the XPLANT macro to define a label L1. Then use the macros XDELETE XINSERT XCHANGE to edit the file. When XPLANT is subsequently used to process this file, specifying L1 will give the original file. This use is similar to the YS and NO edit groups in ELECTRIC. Later version can be created by repeating the procedure.

(b) To dynamically PLANT text strings and SUPPLY files (ie $P, $A and $S in ELECTRIC), use the $P and $ADD XPLANT macros. The PLANT and SUPPLY Xedit macros make this easier.

(c) The ELECTRIC concept of numeric line labels has been extended to allow any string to be used as the label of an edit, so the group edit is no longer needed. For the useful facility of entering one label on the command line and activating many edits, use the $GROUP macro.

(d) It is no longer necessary to have many group edits to allow for all eventualities, XPLANT allows the user to manipulate and test the values of parameters and take appropriate action.

3.1 Labels

The concept of labels in XPLANT is much wider than in ELECTRIC (see Section 2.2), so their use in flagging edits is simple. To define the current label use the XPLANT XEDIT macro. This will also remove any serialisation which exists and change the line length of the file to accommodate edits.

The use of the XPLANT XEDIT macros to change(XCHANGE), delete(XDELETE) or insert(XINSERT) etc.(see section 2.5) will result in these edits being performed conditionally on the label being set when XPLANT is run on the file.

With the ELECTRIC edit file

```
        1 $G LN=   0.  1,DG=ONLY( 1) : DEFAULT
        1 $G LN=   0.  2,DB=ONLY( 5) : DEBUG PRINTING
        5 $I LN=   5.  1 :         WRITE(6,*)A,B,C,D
        5 $I LN=  10.  1 :         WRITE(6,*)I,J
```

DB is used on the command line to activate the edits with label 5.

In an XPLANT file, the lines

```
{DEBUG :        WRITE(6,*)A,B,C,D}
{DEBUG :        WRITE(6,*)I,J}
```

in the appropriate places would have the same effect if DEBUG was set when XPLANT was used.

Instead of having separate labels and parameters as in ELECTRIC, XPLANT allows a parameter to be used as a label. The line

```
{TAPE :/*SETUP {TAPE},R,SL}
```

would only be output if TAPE had a value, in which case its value would appear on the line in place of {TAPE}.

3.2 $A

The $ADD macro will add the contents of a CMS file to the current file.

```
{$ADD MY DATA A 11 10}
```

will add 10 lines starting at line 11 from file MY DATA A to the output file. XPLANT brackets in this file will be interpreted and obeyed unless the additional argument *PLANT=NO is given. The example below shows how a simple file can fetch several other files. Thus several jobs could access the same source or data files without the need for duplication.

```
{$JOBCARD 8 ETC..................}
//STEP1 EXEC F{COMPILER|:H}CLG,PARM.L='NOMAP,OVLY',
//    CPRINT={CPRINT|:YES}
//C.SYSIN DD *
{$ADD EXAMPLE FORTRAN A }
{$ADD SUB1 FORTRAN A L1=ON DIM1=10 }
{$HIDING DEBUG ${$ADD SUB2 FORTRAN A }}
//L.LIB1 DD DSN=ULIB.MYLIB,DISP=SHR
{$ADD OVERLAY STRUCTUR A}
   INCLUDE LIB1(PROG{VERSION|:0})
   ENTRY MAIN
//G.SYSIN DD *
{$ADD CURRENT DATA A}
/*
```

All label settings are passed to the file to be added. The example show three ways of using this fact.

(a) All the current label settings are passed to EXAMPLE FORTRAN A so any additional labels required in the added file should be set on the command line or using the $SET macro before the $ADD macro is called.

(b) In addition to the labels set in the top level file, when SUB1 FORTRAN A is $ADDed L1 will be set 'ON' and DIM1 will have the value 10. These labels only have values inside this bracket so there can be no clash with label names in the top level file.

(c) The $HIDING macro temporarily unsets the label DEBUG so that even if it is set on the command line it will have no effect in SUB2 FORTRAN A

The easiest way to insert $ADD macro calls into a file is to use the SUPPLY Xedit macro(See Section 2.5).

3.3 $C

The ELECTRIC edit command $C prevented an existing edit from being performed even if its label was selected. The XPLANT bracket structure allows several edits in a bracket, only one of which will be performed. So, provided care is taken in providing separate labels for logically separate groups of edits, it should never be necessary to 'cancel' edits. To prevent an edit being performed, do not set the relevant label!

However, in some circumstances it may be easier to stop certain edits being performed than to relabel a complex structure of edits. The method of achieving this depends on the type of edit and the desired result.

(a) To prevent any output from a bracket

EITHER -- conditionally delete the whole bracket

```
To delete   {L1: Some text}
Use       : set arbchar on &
            xplant L2
            XDelete /{&}/
```

This will produce {L2:|:{L1:Some text}}

OR -- add another unit at the beginning of the bracket which will produce no output. eg change {L1:Some text} to {L2:|L1:Some text}

In both of these examples, if L2 has a value the resultant bracket will produce no output even if L1 has a value.

(b) In some cases the whole bracket cannot be deleted because it contains further edits or a default string.

```
The first {L1:new|:old} line
```

When cancelling the edit with label L1 the bracket must still output the default string.

To do this, conditionally change the label L1 to 'NO'.

```
XPLANT L2
XC/L1/NO/
```

This will change the above line to

```
The first {{L2:NO|:L1}:new|:old} line
```

This bracket will output 'new' only if L1 has value and L2 does not. Any other combination of L1 and L2 will return 'old'.

(c) $P edits with a default value can be cancelled as follows.

```
{L1$P VOL USDSK1}
```

should be changed to

```
{L2:USDSK1|L1$P VOL USDSK1}
```

Then if L2 has a value, the default string will be output.

3.4 $D

A line can be conditionally deleted by enclosing it in brackets as below:

```
{L1:|: The line of text }
```

Several lines may be deleted at once

```
{L1:|: One line of text
Another line
Yet another line. }
```

In both of these examples nothing would be output if label L1 has a value.

These edits are easily applied to existing lines of text by using the XDELETE xedit macro (see section 2.5).

Note that conditional deletion of a line is identical to insertion of a line conditional on a label NOT being set. So the line

```
{{$NOT L1}: The line of text.}
```

would have the same effect as the first example.

3.5 $E

To conditionally change a string of text a bracket is needed which will output one string if a label has a value and the original string if it has not. This is achieved by using the XChange xedit macro. Thus, given the line

```
This is an old line of text.
```

First set the label to L1

```
XPLANT L1
```

Then

```
XC/n old/ new/
```

will produce

```
This is a{L1: new|:n old} line of text.
```

which will output

```
This is a new line of text.
```

if L1 has a value and the original line if it does not.

3.6 $G

The $GROUP macro sets a number of labels to the value 'ON'. This macro allows the logical grouping of edits and helps in the organisation of complicated edit structures. If the macro call is made conditional on another label then only one label need be set on the command line and the others associated with it can be set inside the file.

XPLANT allows the user to take the logical OR (by using several labels) or the logical AND (see Section 4.3) of labels, so it is not as necessary, as in ELECTRIC, to have a number of edit labels which are selected by a 'Group edit'.

```
      {G1 $GROUP L1 L2 L3}
```

will set the labels L1,L2,L3 if G1 has a value.

To UNSET a group of labels (the opposite of $GROUP), use the UNSETS macro.

```
      {G3 $UNSETS L1 L2 L3}
```

This will remove any values that L1, L2, L3 may have had.

3.6.1 Examples

The following file has three labelled edits in two logical groups.

```
      {G1 $GROUP L1 L2}
      {G2 $GROUP L1 L3}
      {L1:     The first line.}
      {L2:     The second line.}
      {L3:     The third line.}
```

So if G1 has a value, then when the file is XPLANTed ,

```
      The first line.
      The second line.
```

would be output. Similarly if G2 had a value then

```
      The first line.
      The third line.
```

would be output.

Note that, unlike ELECTRIC edits, XPLANT brackets may have several labels so the same effect could be achieved more simply by

```
      {G1 G2:     The first line.}
      {G1:     The second line.}
      {G2:     The third line.}
```

or even,

```
      {G1 G2:     The first line.
      {G1:     The second line.}
      {G2:     The third line.}}
```

This last is more efficient because if neither G1 nor G2 is set then the second and third brackets are not processed at all. In all of these examples if both G1 and G2 have values then all three lines will be output.

If a third group is defined which always removes label L1.

```
{G1 $GROUP L1 L2}
{G2 $GROUP L1 L3}
{G3 $UNSETS L1}
{L1:    The first line.}
{L2:    The second line.}
{L3:    The third line.}
```

Then using XPLANT with G1, G2 and G3 set will result in

```
The second line.
The third line.
```

being output.

3.7 $I

Lines may be conditionally inserted in a file by using the method shown in Section 3.6 . The easiest way to achieve this construction is to use the XEDIT macro XInsert. This is used in the same way as the XEDIT command Insert but the resultant edits are conditional on a label being set. This label is set by the XPLANT sub-command and remains the same until reset. To make existing lines in a file only be output if a certain label is set, use the ENCLOSE macro. This will enclose the lines in brackets and make their insertion conditional on the current label being set.

Below is a record of a short editing session to conditionally insert some lines in a file.

```
X SHORT FILE
EDITING FILE: SHORT FILE A1
XEDIT:
/if
     IF(I.EQ.1)THEN
t6
     IF(I.EQ.1)THEN
          J=0
          K=0
          ELSE J=J+1
          K=K+1
     ENDIF
u3
          K=0
xplant output
xi       write(6,'(1x,10f10.4)')x
2
          K=K+1
xi       write(6,101)x(i),x(j),x(k)
xi       write(6,102)y(i)
d
     ENDIF
xi
input mode:
```

```
101  format(5x,' x cords   ',3f10.4)
102  format(5x,'  height   ',f10.4)
XEDIT:
-/if(
     IF(I.EQ.1)THEN
t11
     IF(I.EQ.1)THEN
          J=0
          K=0
{OUTPUT:    WRITE(6,'(1X,10F10.4)')X}
          ELSE J=J+1
          K=K+1
{OUTPUT:    WRITE(6,101)X(I),X(J),X(K)}
{OUTPUT:    WRITE(6,102)Y(I)}
     ENDIF
{OUTPUT:101  FORMAT(5X,' X CORDS   ',3F10.4)
102  FORMAT(5X,'  HEIGHT   ',F10.4)}
```

3.8 $P

A simple replacement for $P is provided by using the fact that a bracket containing only a label returns the value of the label. Using the desired keyword as a label will plant its value in the file.

```
/*SETUP  {TAPE},R,SL,BLP
```

will be output as

```
/*SETUP 912345,R,BLP
```

if TAPE has the value 912345.

To make this conditional on label 1 being set replace {TAPE} by {1:{TAPE}}.

To allow a default value

```
/*SETUP  {TAPE|:900001},R,BLP
```

will return the value of TAPE if it has one, otherwise it will return 900001.

A unit is processed from left to right until a label is found with a value. This enables the user to give one of several different keywords in the same place.

```
/*SETUP  {TAPE T VOL VOLSER},R,BLP
```

would replace the bracket by the first of the four labels to have a value. Thus the keyword planted could have abbreviations or alternative spellings.

The $PLANT (or $P) macro (see section 2.4) provides a more powerful facility which will optionally check the value of the keyword and/or prompt for a value and/or allow a default value.

```
/* SETUP {$P VOL % VOLSER 'Give the Volume name '}
```

If VOL is already set its value will be checked to be of type VOLSER If it is not , a prompt ' Give the Volume name ' will be issued. If the entered value is correct, then VOL will be set to it and the bracket replaced by the value of VOL.

3.9 $R

To replace a line one constructs a bracket that outputs the new lines if the label is set and outputs the old one if the label is not set. For example if the label is set to L1 by,

```
xplant L1
```

and the current line is

```
The old line.
```

then using the xreplace xedit macro thus

```
xreplace The completely new line
```

will result in

```
{l1:The completely new line|
:The old line.}
```

This will output 'The completely new line' if L1 is set and 'The old line' if L1 is not set.

If XREPLACE is called with no arguments, then Xedit will enter input mode thus allowing several lines to replace one. So in the example above

```
XREPLACE
```

followed by

```
The first new line.
The second new line.
<null>
```

will result in

```
{L1:The first new line.
The second new line.|
:first old line.}
```

4. EXTENSIONS AND ALTERNATIVES.

4.1 The CMS UPDATE Command

The CMS product as delivered by IBM also contains a method of delayed editing and of storing multiple versions of a file. This is the UPDATE command. If the XEDIT editor is invoked with the UPDATE option then, as in ELECTRIC, any edit sub-commands given are stored in a separate file and not performed on the file being edited although the file displayed at the terminal will appear to have changed. This means that, unlike ELECTRIC, the user can see the edits he has made as he makes them.

If the UPDATE command is used on the original file then a new file can be created which will contain the original file with the edits performed. The user can specify the name of the 'edit' file so multiple versions of a file require multiple electric files, unlike ELECTRIC. A number of these edit files may be applied sequentially to the same source file allowing edit structures similar to that provided by the labels in ELECTRIC.
Although this method does not allow dynamic planting of strings or any of the other powerful features of XPLANT, it is supported by IBM and might be preferable for the user who frequently transfers files to other CMS installations. For more information on the UPDATE command, see the relevant IBM manual[6]

4.2 Using Parameters and Functions as Macro Arguments.

In most of the examples in the previous sections, XPLANT macros have been called with arguments which are simple text strings.

```
{ $ADD EXAMPLE FORTRAN A }
```

Here $ADD has three arguments EXAMPLE, FORTRAN and A. Any of these arguments may contain brackets. These will be resolved before the macro is called. So

```
{$ADD {FORT1|:EXAMPLE} FORTRAN A }     changes the filename

{$ADD EXAMPLE FORTRAN {MODE} }         changes the filemode

{$ADD EXAMPLE{NUMBER} FORTRAN A }      changes part of the filename
```

An argument may also be a macro call itself.

```
{$ADD {$P FORT1 % FN 'Fortran Filename ?'} FORTRAN A }
```

will check the value of FORT1 to ensure that it is a valid CMS filename and prompt the user with the message 'Fortran Filename ?" if it is missing or incorrect. This gives the user greater flexibility in calling macros. In the above examples, FORT1,MODE or NUMBER could be supplied when the file is XPLANTed effecting the choice of file to be $ADDED.

Users should differentiate between using the name of a parameter and using its value.

```
{$JOBCARD % {TAPE}}
{/*SETUP { $P TAPE % VOLSER }
```

On the first line the value of TAPE is passed to the $JOBCARD macro as the jobname. On the second the text string TAPE is used by $P. For these 2 lines

```
XPLANT TEST (TERM(TAPE 901234
```

would produce

```
OS jobname is "PY901234"
//PY901234 JOB (1234,PY,,,,,,,),
// 'VM/JCG/PH-33'
//* Command line:XP TESTIT ( TERM ( TAPE 901234
//* $JOBCARD called from "TESTIT   XPLANT   A1 1"
//* Submitted from RAL CMS userid "JCG" on 11/18/82 14:54:52
/*SETUP 901234
```

4.3 Macro calls as Labels

Most of the previous examples have used labels to make conditional calls to XPLANT macros. If a parameter is used as a label then the bracket will be 'active' if the parameter has a value and ignored otherwise. If the label is a bracket then this will be resolved when the label is evaluated, so the label will be whatever is output by the bracket. Several XPLANT macros have been provided which return different values depending on the values of their arguments so they can be used as labels to test the values of parameters, not merely whether or not the parameter has a value.

For example, the macro $EQUAL has two arguments. These arguments are compared and if they are equal then the string 'YES' will be returned. If they are not equal then 'NO' will be returned. When XPLANT processing starts, the parameter YES is set on while NO is unset.

If the parameters X and TEST1 have the same value then the bracket

```
{$EQUAL {X} {TEST1}}
```

will be replaced by YES. This means that

```
{{$EQUAL {X} {TEST1}} $TYPE X is OK }
```

is equivalent to

```
{ YES $TYPE X is OK }
```

and as YES has a value, then the $TYPE macro will be executed.

Similarly {$LESS A B} will return YES if A has a value numerically less than that of B.

If in the following bracket , A=5 and B=1

```
{{$LESS A B} $SET L {A}${A}|{$LESS B A}$SET L {B}${B}|$TYPE A=B}
```

the bracket will be equivalent to

```
{ NO $SET L 5 $ 5| YES $SET L 1 $ 1| $TYPE A=B}
```

As YES has a value and NO does not, then the first unit will be skipped and the second unit executed. L would be set to '1' and '1' would be output.

Similar use can be made of :-

(a) $CHECK To check the type of a value.

(b) $MORE First argument numerically greater than the second.

(c) $AND Logical AND of the two arguments.

So the sequence

```
        {$SET FTAPE 980001}{$SET LTAPE 980999}
        {{TAPE $CHECK TAPE VOLSER}:
        {{$LESS TAPE FTAPE {{$MORE TAPE LTAPE}$TYPE {TAPE} is not one of your
```

will first check that TAPE is a proper Volume Serial Number, then it will complain if TAPE is outside the range FTAPE to LTAPE.

Note that there is no need for a $OR macro because of the way that Xplant processes labels. If more than one label is given then Xplant will evaluate them from left to right searching for one with a value. This is equivalent to a logical OR of all the labels given.

4.4 User Defined Macros

In order to avoid repetition of sequences of text and brackets many times in a file, XPLANT allows users to define their own macros. These contain a number of lines which are processed every time the macro is called.

The arguments for the $MACDEF function are

```
{$MACDEF macro-name first-argname second-argname . . .$first-line
 second-line
 .
 .
 last-line}
```

This macro defines a macro called macro-name. The positional arguments are first-argname, second-argname, etc. The body of the macro is the lines following the second $.

For example, the following Fortran file defines three macros. The first inserts a block of comment cards with an imbedded text string. The other two insert common blocks with numbers imbedded.

```
{$MACDEF CREDIT *A  $
C
C                   ROUTINE WRITTEN BY
C                       {*A}
C                   USER SUPPORT GROUP
C             ----  PLAGIARISTS BEWARE.   ----
C}
{$MACDEF CB1 $
      PARAMETER (LEN=100)
      COMMON / BLOCK1 / A(LEN),B(LEN),C(LEN)
C}
{$MACDEF CB2 $
      PARAMETER (LEN2=1000)
      COMMON / BLOCK2 / X(LEN2),Y(LEN2,10),Z(16,16)
C}
      PROGRAM MAIN
{$CREDIT 'JOHN GORDON'}
{$CB1}
{$CB2}
      CALL SUB1
      CALL SUB2
      END
      SUBROUTINE SUB1
{$CREDIT 'JOHN GORDON'}
{$CB1}
      CALL SUB3
      END
      SUBROUTINE SUB2
{$CREDIT 'JOHN GORDON'}
{$CB2}
      CALL SUB4
      END
```

```
XPLANT TESTIT XPLANT A TESTIT FORTRAN A
```

```
      PROGRAM MAIN
C
C                   ROUTINE WRITTEN BY
C                       JOHN GORDON
C                   USER SUPPORT GROUP
C             ----  PLAGIARISTS BEWARE.   ----
C
      PARAMETER (LEN=100)
      COMMON / BLOCK1 / A(LEN),B(LEN),C(LEN)
C
      PARAMETER (LEN2=1000)
      COMMON / BLOCK2 / X(LEN2),Y(LEN2,10),Z(16,16)
C
      CALL SUB1
      CALL SUB2
      END
      SUBROUTINE SUB1
C
C                   ROUTINE WRITTEN BY
C                       JOHN GORDON
C                   USER SUPPORT GROUP
C             ----  PLAGIARISTS BEWARE.   ----
C
```

```
      PARAMETER (LEN=100)
      COMMON / BLOCK1 / A(LEN),B(LEN),C(LEN)
C
      CALL SUB3
      END
      SUBROUTINE SUB2
C
C                    ROUTINE WRITTEN BY
C                        JOHN GORDON
C                    USER SUPPORT GROUP
C             ----   PLAGIARISTS BEWARE.   ----
C
      PARAMETER (LEN2=1000)
      COMMON / BLOCK2 / X(LEN2),Y(LEN2,10),Z(16,16)
C
      CALL SUB4
      END
```

Macros are particularly useful in this case for they ensure that the text is identical each time it is repeated so the updating of common blocks structures is made easier.

Appendix A

XPLANT_FUNCTIONS.

This is a list of intrinsic functions and macros available in XPLANT, arranged in groups according to function.

For more detail, there is an XPLANT MENU, and separate HELP files for each function, e.g. HELP XPLANT $ADD. When calling any of these functions, note that the parameters may be given as either positional or keyword=value pairs. Parameters for controlling the system-provided functions begin with "*" to prevent confusion with user-defined variables.

Common Functions for JCL and planting:

$JOBCARD	generates a /*PRIORITY and HASP jobcard fully planted with checking.
$P	obtains a value from the command line or the terminal with value-type checking (equivalent of ELECTRIC $P), can also prompt for value.
$PLIST	obtains a value from the command line or the terminal (like $P above) but checks value against list instead of type.

CMS and Spool File Handling and terminal I/O:

$ADD	includes all or part of a specified file - equivalent of the $A and $S in ELECTRIC
$LOOPA	prompts for list of CMS fileids to be $ADDed
$ERASE	erases a CMS file
$STATE	obtains information relating to a CMS file
$FILESTAT	useful for coping with the output from $STATE
$OUTFILE	selects the output file name
$PRINT	selects the device address for printer output
$PUNCH	selects the device address for punch output
$PAGE	throw a page on the current printer output stream
$READ	reads a line from the terminal
$LOOPI	prompts for input lines to be optionally checked and justified before being output
$TYPE	types a line on the terminal
$STACK	stack a line in the CMS console stack
$LISTVAL	stack selected XPLANT variable names
$TERMINA	turns on and off output to the terminal

String Manipulation:

$CHANGE	substitutes a substring of a string for another string
$COMLINE	returns the invoking command line
$INDEX	finds the character offset of one string in another
$POSITION	returns index of word from a list of words (like &POSITION in EXEC2)

$JUSTIFY	generates fixed width fields with text justified to left or right
$LENGTH	finds the length of a string
$LOWER	translates a string to lower case
$UPPER	translates a string to upper case
$OPTION	reads a token from the command line
$QSET	gets the value of a CMS set variable (QSET)
$WORD	returns the Nth word in a list of words
$FIRST	returns the first word in a list of words
$REST	returns all but the first word of a list of words
$LAST	returns the last word in a list of words
$SUBSTR	gets a specific substring of a string

Functions useful as labels:

$CHECK	checks that a value is of a specified type e.g DDname - result is useful as a label
$CKLIST	checks that a value is one of list of values.
$AND	logical AND of two variables to make a label
$NOT	logical NOT of a variable to make a label
$EQUAL	compares two values - used as a label
$LESS	arithmetically compares two numbers - used as a label
$MORE	arithmetically compares two numbers - used as a label

Control:

$RETURN	returns out of a file that is being $ADDed
$STOP	stops XPLANT (same as EXIT)
$EXIT	stops XPLANT (same as STOP)
$QUIT	aborts XPLANT
$HIDING	evaluate data with a variable temporarily unset
$INSERT	performs no special function but is useful to process the text after the second "$" and set the parameters - useful for user macros
$TRACE	alters the level of XPLANT trace and dump
$SETSYNTAX	changes definition of special characters
$MAP	apply a function to lists of arguments many times

Settings variables globally:

$SET	sets the value of a variable globally throughout the program
$SETFILE	sets variables from data in a file
$SETVARS	sets multiple variables to multiple values
$UNSET	unsets a $SET variable
$UNSETS	unsets a number of variables
$GROUP	sets many variables to the value ON - useful for setting ON several labels together $UNSETS unsets many variables whether they have been $SET or not.

Miscellaneous, date, CP/CMS commands, addition, subtraction etc:

$CMS	issues a CMS command
$CP	issues a CP command
$DATE	gets the date and time
$MINUS	subtract two numbers

$PLUS	add two numbers
$TIMES	multiply two numbers
$QUOTIENT	quotient of two numbers
$REMAINDER	remainder of two numbers
$SETVALUE	find the 'SET' value of a variable
$USERID	obtains the username of the current XPLANT user
$VALUE	obtains the value of a variable
$WHERE	finds the current XPLANT filename and record number
$MACDEF	define a macro
$MACLIB	attach a library of macro definitions

REFERENCES

[1] XPLANT Introduction and Reference Manual, D M Asbury RL-83-???

[2] CMS users guide SC19-6210

[3] RAL VM Reference Manual RL-79-083

[4] Introduction to CMS at RAL RL-80-008

[5] For information on these courses contact the Program Advisory Office at Rutherford Appleton Laboratory.

[6] IBM VM/SP CMS Command and Macro Reference SC19-6209