IBM

Application Program

GH20-0596-1

System/360 Problem Language Analyzer (PLAN) (OS) Operations Manual

Program Number 360A-CX-27X

This manual is intended to assist users in the implementation and execution of PLAN jobs using OS/360. It includes specifications pertinent to only the System/360 OS version of PLAN. Sections of special interest pertain to debugging, diagnostics, and abnormal termination. This manual should be read before attempting any computer operations.

Second Edition (January 1970)

This edition is a major revision obsoleting H20-0596-0.

This edition applies to Version 1, Modification Level 1 of System/360 Problem Language Analyzer (PLAN) (OS) (360A-CX-27X) and to all subsequent versions and modifications until otherwise indicated in new editions or Technical Newsletters.

Changes are continually made to the specifications herein. Therefore, before using this publication, consult the latest System/360 SRL Newsletter (GN20-0360) for the editions that are applicable and current.

Copies of this and other IBM publications can be obtained through IBM branch offices.

A form has been provided at the back of this publication for readers' comments. If this form has been removed, address comments to: IBM Corporation, Technical Publications Department,
112 East Post Road, White Plains, N. Y. 10601

The Problem Language Analyzer (PLAN) under OS/360 provides the user with an efficient means of implementing and using problem-oriented languages. This manual is intended to assist users in the implementation and execution of PLAN jobs using OS/360.

This manual contains sections that provide the user with the following:

1. Examples of PLAN job processing
2. Descriptions of the diagnostic messages produced during execution of a PLAN job
3. Descriptions and explanations for the preparation of programs to be executed under PLAN that are written in languages other than FORTRAN

The user of this manual should be familiar with the following publications:

Problem Language Analyzer (PLAN) Program Description Manual (H20-0594)

IBM System/360 Operating System: Job Control Language (C28-6539)

IBM System/360 Operating System: Linkage Editor (C28-6538)

IBM System/360 Operating System: FORTRAN IV Language (C28-6515)

The user should also be familiar with one of the following:

OS/360 FORTRAN IV (E) Programmer's Guide (C28-6603)

OS/360 FORTRAN IV (G) Programmer's Guide (C28-6639)

OS/360 FORTRAN IV (H) Programmer's Guide (C28-6602)

GENERAL

The IBM Operating System/360 consists of a control program and processing programs. The control program supervises the execution of all processing programs, such as the PLAN system monitor. Therefore, to execute a PLAN job, the user must first communicate with the operating system and the medium of communication between the user and the operating system is JOB CONTROL language.

To the operating system, a JOB consists of executing one or more job steps. In order to execute any job under OS/360, the user must first describe to the system the required job steps and the data sets to be processed by those steps. He defines a job to the operating system by using a JOB statement. A job step is defined by using an EXEC statement, and a data set is defined by using a DD statement.

The PLAN system is similar to OS/360 in that it supervises the execution of other problem program modules, and must have available a description of its job requirements before it can execute a PLAN job. The medium of communication with the PLAN system is through PLAN phrases and commands.

A PLAN phrase is a definition of a PLAN job step. Each such definition normally contains (1) a list of problem programs to be executed, and (2) a list of input parameters and/or constants.

A PLAN command is a statement that causes the PLAN system to invoke or execute a certain phrase description.

PLAN SYSTEM DESCRIPTION

The PLAN system monitor has three main elements: (1) the interpreter, (2) the executor, and (3) the phrase dictionary. Figure 1 is a logical schematic of the PLAN system.

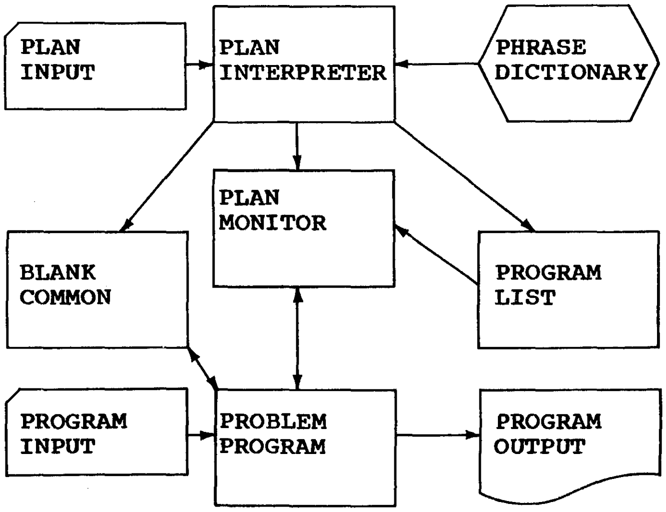


Figure 1. PLAN system

Input data is read by an interpreter. Based on a PLAN job definition in the phrase dictionary, a list of programs to be executed and a blank common area are prepared.

The executor then loads and executes the programs named in the program list. When the program list is exhausted, the executor returns control to the interpreter and the cycle repeats itself.

PLAN SYSTEM REQUIREMENTS

In order to execute a job, the PLAN system must have available the facilities listed below:

1. An input device from which PLAN commands can be accepted
2. An output device through which PLAN may communicate with the user
3. A phrase dictionary that contains the PLAN job definitions
4. A library of executable programs

To provide these things to PLAN, the user must execute a three-step process: (1) generate the required programs for the job, (2) define the job requirements by adding phrases to the PLAN phrase dictionary, and (3) execute the necessary PLAN commands to run the job. Figure 2 is a logical schematic of this process.

GENERATING THE EXECUTABLE PROGRAMS

In order to generate executable PLAN modules, the user must process his FORTRAN source code through two OS job steps. The first is the compile step in which an OS FORTRAN compiler produces an object deck from the source deck. The second is the link-edit processing step which converts the object module into an executable load module. Sample 1 illustrates a job stream to create an executable module named M0107.

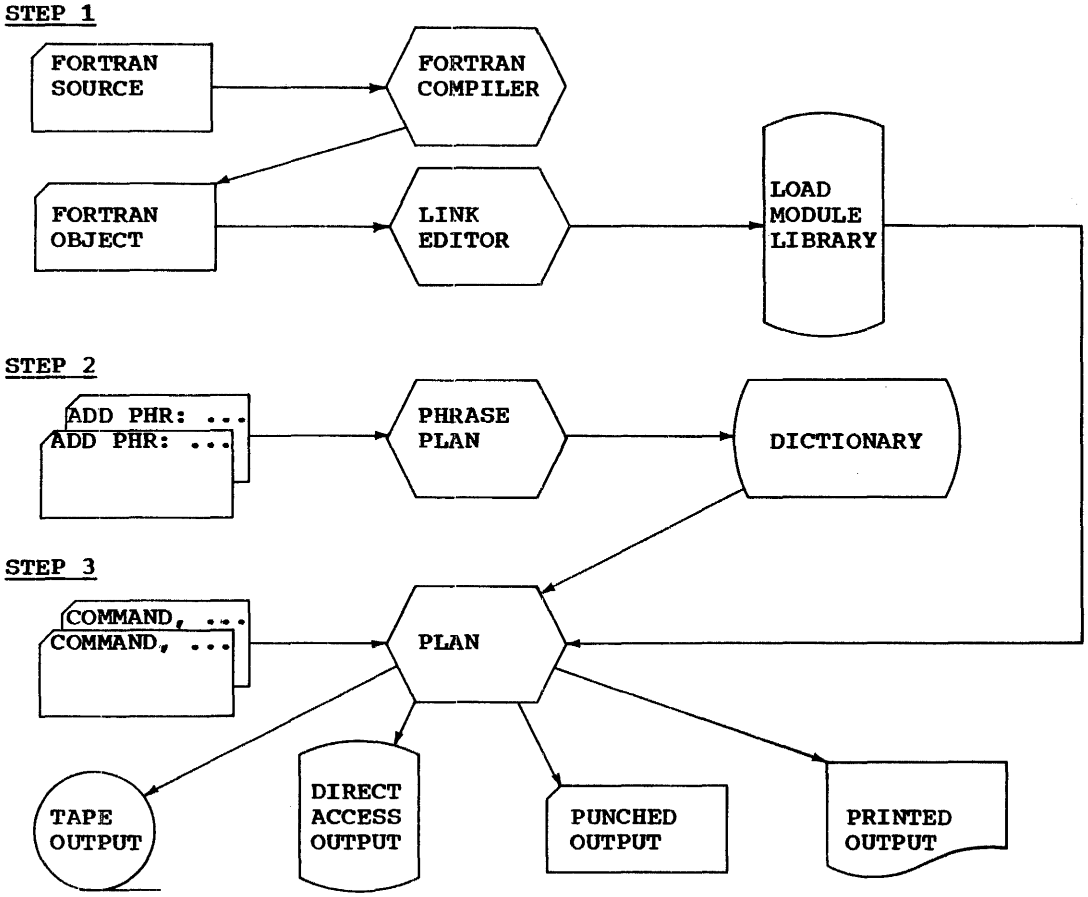


Figure 2. Necessary steps for PLAN execution

```
//FORTRAN JOB  84803,'JOE E. JONES',MSGLEVEL=1                          00010
//COMP    EXEC PGM=IEJFAAA0,PARM='ADJUST,NAME=M0107'                    00020
//SYSPRINT DD  SYSOUT=A                                                 00030
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(2,1))                             00040
//SYSUT2   DD  UNIT=SYSDA,SPACE=(CYL,(2,1))                             00050
//SYSLIN   DD  DSNAME=&LOADSET,UNIT=SYSSQ,SPACE=(400,(200,50)),       X 00060
//             DISP=(NEW,PASS)                                          00070
//SYSIN    DD  *                                                        00080
      COMMON L (625), LS (15) , MA (255) , NMA (255)                    00090
         •                                                              00100
         •                                                              00110
      FORTRAN SOURCE STATEMENTS                                         00120
         •                                                              00130
         •                                                              00140
      END                                                               00150
/*                                                                      00160
//LINK    EXEC PGM=IEWL,PARM=(LIST,LET),COND=(4,LT,COMP)                00170
//SYSPRINT DD  SYSOUT=A                                                 00180
//SYSLMOD  DD  DSNAME=MYLIB,UNIT=2311,VOLUME=SER=MY2311,DISP=OLD        00190
//SYSLIB   DD  DSNAME=SYS1.FORTLIB,DISP=OLD                             00200
//         DD  DSNAME=PLAN.SUBLIB,UNIT=2311,VOLUME=SER=PLANPK,DISP=OLD  00210
//SYSUT1   DD  UNIT=SYSDA,SPACE=(CYL,(2,1))                             00220
//SYSLIN   DD  DSNAME=&LOADSET,DISP=(OLD,DELETE)                        00230
//         DD   *                                                       00240
  NAME  M0107(R)                                                        00250
/*                                                                      00260
```

Sample 1. Job stream to create M0107

EXPLANATION OF COMPILE AND LINK-EDIT STEPS

Card 00010 is a valid job card.

Card 00020 is the execute card required to execute the OS FORTRAN E compiler. The program to be executed is IEJFAAA0. The PARM operand indicates that the ADJUST option is to be used and the NAME of the output object module is M0107.

Card 00030 is a DD card specifying the data set for printed output from the compiler.

Cards 00040 and 00050 are DD cards specifying utility work data sets for the FORTRAN E compiler.

Cards 00060 and 00070 are DD cards specifying the data set to receive the output object module.

Card 00080 is a DD card specifying the data set that contains the source card input. In this case, it is the input stream.

Cards 00090 through 00150 are the FORTRAN source cards.

Card 00160 is an OS job-step delimiter.

At the completion of this job step, the FORTRAN E compiler will have produced an object deck on the data set named &LOADSET and a program listing on the output device named as the CLASS A WRITER.

Card 00170 is an execute card for the link-edit step. The program to be executed is IEWL, the OS link-editor. The COND field specifies that the link-edit step is not to be run if the compile step fails.

Card 00180 is a DD card specifying a data set for printed output from the link-editor.

Card 00190 is the DD card specifying a partitioned data set to receive the output load module.

Cards 00200 and 00210 are DD cards specifying the subroutine libraries to be used to resolve subroutine calls in the source program. SYS1.FORTLIB is the standard OS FORTRAN subroutine library, and PLAN.SUBLIB contains the PLAN subroutines.

Card 0220 is a DD card specifying a utility work data set for the link editor.

Cards 00230 and 00240 are DD cards specifying the input data set for the link-editor. (Note: This same data set was used for the output of an object module in the compile step.) The input data set is concatenated with the input stream which contains link-edit control cards.

Card 00250 is the link-edit control card specifying the name of the output load module.

Card 00260 is an OS job-step delimiter.

At the completion of this step, the data set MYLIB will contain the executable load module M0107.

The user should refer to the appropriate FORTRAN programmer's guide for other examples of usage of FORTRAN compilers and link-edit facilities of the operating system.

ADDING PHRASES TO THE DICTIONARY

PLAN job requirements are defined in the PLAN phrase dictionary. This dictionary is a data set on a direct access device. Before executing any PLAN jobs, the user must define this data set and add his phrases to the dictionary. Sample 2 illustrates a job stream to create the dictionary data sets and to add a phrase.

```
//PLANINIT JOB 84803,'JOE E. JONES',MSGLEVEL=1                              00010
//STEP     EXEC PGM=DFJPLAN                                                 00020
//PLOUT100 DD   SYSOUT=A                                                    00030
//PLANLIB  DD   DSNAME=PLAN.MODLIB,VOLUME=SER=PLANPK,UNIT=2311,DISP=OLD     00040
//PLSYSTAB DD   DSNAME=PFILE,UNIT=2311,VOLUME=SER=MY2311,                   00050
//              DISP=(NEW,KEEP),SPACE=(CYL,(5))                             00060
//PLINP001 DD   *                                                           00070
ADD PHRASE: PLAN JOB,PRO'M0107',LEVEL 1,I(1) RUNTYPE 0;                     00080
/*                                                                          00090
```

Sample 2. Job stream creating dictionary data sets and adding a phrase

ADD PHRASE STEP EXPLANATION

Card 00010 is a valid job card.

Card 00020 is the execute card. The program to be executed is DFJPLAN, the PLAN system monitor.

Card 00030 is a DD card defining the PLAN output device to be a SYSOUT writer class A.

Card 00040 is a DD card defining the PLAN library PDS, which is a data set called PLAN.MODLIB. PLAN obtains all executable modules from this library.

Cards 00050 and 00060 are the DD cards defining the PLAN system dictionary data set. The disposition of (NEW,KEEP) specifies that the data set is to be formatted by the PLAN system and retained for use in subsequent PLAN executions.

Card 00070 is a DD card defining the PLAN input stream.

Card 00080 is the ADD PHRASE card shown adding the phrase PLAN JOB.

Card 00090 is the OS job-step delimiter.

At the conclusion of this job step, PLAN will have created a phrase dictionary data set with the name PFILE on a 2311 with the serial number MY2311. The phrase PLAN JOB will be added to this dictionary for use by subsequent PLAN executions.

EXECUTING A PLAN JOB

After the user has generated the required modules and entered his phrases into the PLAN phrase dictionary he may execute his PLAN job. Sample 3 illustrates a job stream for job execution.

```
//PLANJOB JOB  84803,'JOE E. JONES',MSGLEVEL=1                              00010
//PLAN     EXEC PGM=DFJPLAN,PARM='TRACE'                                    00020
//PLSYSTAB DD   DSNAME=PFILE,UNIT=2311,VOLUME=SER=MY2311,DISP=OLD           00030
//PLANLIB  DD   DSNAME=PLAN.MODLIB,UNIT=2311,VOLUME=SER=PLANPK,DISP=OLD     00040
//         DD   DSNAME=MYLIB,UNIT=2311,VOLUME=SER=MY2311,DISP=OLD           00050
//PLOUT100 DD   SYSOUT=A                                                    00060
//PLINP001 DD   *                                                           00070
 PLAN JOB,RUNTYPE = 3;                                                      00080
/*                                                                          00090
```

Sample 3. Job execution

PLAN JOB STEP EXPLANATION

Card 00010 is a valid job card.

Card 00020 is the execute card for the PLAN step. The program is DFJPLAN, the PLAN system monitor. The PARM 'TRACE' invokes the PLAN tracing facility.

Card 00030 is the DD card defining the PLAN system phrase dictionary data sets. (Note: This same data set was used when the phrase PLAN JOB was added to the dictionary in Sample 2.)

Cards 00040 and 00050 are the DD cards defining the PLAN load module library. (Note: MYLIB, which contains the user module M0107, is concatenated to PLAN.MODLIB which contains the PLAN system load modules.)

Card 00060 is the DD card specifying a data set for PLAN printed output.

Card 00070 is the DD card specifying a data set for PLAN input.

Card 00080 is the PLAN command that invokes PLAN JOB, which was added to the dictionary.

Card 00090 is an OS job-step delimiter.

While executing this job step, the PLAN system monitor searches the PLAN phrase dictionary in the data set PFILE for the phrase PLANJOB. Finding it initializes a blank common area based on the input parameters in the PLAN command. Then the program M0107 is loaded from the library MYLIB and entered for execution.

OS PLAN PROCESSING

To execute the PLAN system under OS/360, the user must prepare the necessary JCL (Job Control Language) statements. These are (1) a JOB statement, (2) an EXECUTE statement, (3) DD statements, as required, and (4) an OS standard delimiter. Figure 3 shows the PLAN DDNAME function and device requirements.

DDNAME	FUNCTION	DEVICE
PLSYSTAB	PLAN WORK AREA AND PHRASE DICTIONARY	*INTERMEDIATE STORAGE
PLANLIB	PDS CONTAINING LOAD MODULES TO BE EXECUTED	*INTERMEDIATE STORAGE
PLINPxxx	INITIAL PLAN INPUT	*CARD READER *MAGNETIC TAPE *INTERMEDIATE STORAGE
PLOUTxxx	INITIAL PLAN OUTPUT	*PRINTER *CARD PUNCH *MAGNETIC TAPE *INTERMEDIATE STORAGE
PLMANFIL	MANAGED ARRAY SAVE FILE	*INTERMEDIATE STORAGE
PLCHKPT	CHECKPOINT FILE	*INTERMEDIATE STORAGE
PLANDRVx	PLAN DYNAMIC DRIVE	*INTERMEDIATE STORAGE
PLFSynnn	USER PERMANENT DATA SETS	*INTERMEDIATE STORAGE
PLSEQxxx	SEQUENTIAL INPUT OR OUTPUT	*CARD READER *TAPE *PRINTER *CARD PUNCH

Figure 3. PLAN DDNAME and device requirements

JOB STATEMENT

A valid JOB card must be supplied. Installation standards where the job is run determine the required JOB card parameters.

EXEC STATEMENT

The name of the PLAN system monitor is DFJPLAN; the operands of the PGM keyword must be DFJPLAN.

PARM FIELDS

There are five valid operands of the PARM keyword: PGAR, NFS, TRACE, NOLIST, and PHRAS.

PGAR specifies the length of the PLAN PROGRAM COMMON area. It is coded as PGAR=nnn, where nnn specifies the number of contiguous 1024-byte blocks to be reserved. If this operand is omitted, the size of the PLAN PROGRAM COMMON area is 66 percent of the region or partition.

NFS specifies the length of the nonmanaged OS free storage area. It is coded as NFS=nnn, where nnn is the number of contiguous 1024-byte blocks to be reserved. If this operand is omitted, the size of the nonmanaged free storage area is zero.

TRACE specifies that the name and entry point of all programs loaded and entered for execution be listed on the PLAN output device. (See "PLAN Trace Facility".)

NOLIST specifies that the normal 80-80 list of the PLAN input stream be suppressed.

PHRAS specifies that all commands executed by the PLAN system be listed on the PLAN output device.

The following examples show valid uses of the PARM keyword:

```
//STP EXEC PGM=DFJPLAN,PARM='TRACE'

//STP EXEC PGM=DFJPLAN,PARM='PGAR=60,      X
//          NFS=20'

//STP EXEC PGM=DFJPLAN,PARM='TRACE,NOLIST'
```

JOBLIB DD STATEMENT

This is an optional DD statement. OS/360 PLAN requires the following modules to be available to begin execution: (1) DFJPLAN, (2) DFJLODER, (3) DFJTRACE.

If these modules are not in SYS1.LINKLIB or the LINKAC-RAM area, the user must provide a suitable JOBLIB DD statement that describes a PDS that does contain them.

REQUIRED DD STATEMENTS

The DD statements listed below are required for PLAN execution. If any are missing, execution of PLAN is suppressed.

PLSYSTAB

This DD statement defines the PLAN PFILE data set that contains the system tables and the phrase dictionary. If the disposition is NEW, this file is formatted and the phrase 'ADD PHRASE' is added to the dictionary. The following examples show PLSYSTAB DD statements:

```
//PLSYSTAB DD DSNAME=PFILE,UNIT=2311,        X
//             VOLUME=SER=MY2311,DISP=OLD

//PLSYSTAB DD DSNAME=PFILEA,DISP=OLD

//PLSYSTAB DD DSNAME=PFILEB,UNIT=2311,       X
//             VOLUME=SER=MY2311,            X
//             DISP=(NEW,KEEP),              X
//             SPACE=(CYL,(2))
```

The first example shows the use of a data set named PFILE that already exists on a 2311, serial number MY2311. This data set is to be retained at the end of the PLAN execution.

The second example shows the use of cataloged data sets named PFILEA.

The third example shows the creation of a data set named PFILEB on a 2311, serial number MY2311. Two cylinders on the pack will be allocated, and the data set is to be retained at the end of the PLAN execution. Since the disposition is NEW, the file will be formatted by the PLAN system. It is the user's responsibility to add the standard PLAN commands to his dictionary. (Note: For new data sets the allocation must be at least 14 records. The maximum required allocation is 268 records. The block size for PLSYSTAB is 512.)

PLINPxxx

This DD statement defines a sequential PLAN input data set to be read using the PLAN unit record subroutines. The xxx is a three-digit number equivalent to the NOD parameter in the subroutine call parameter list. The PLINPxxx DD statement will be used as the initial PLAN input device. Only one PLINPxxx DD card is allowed per PLAN job step. The following examples show PLINPxxx DD statements:

```
//PLINP001 DD  *

//PLINP006 DD DSNAME=MYFILE,UNIT=2311,       X
//             VOLUME=SER=MY2311,DISP=OLD,   X
//             DCB=(RECFM=FB,LRECL=80,       X
//             BLKSIZE=400)
```

The first example shows the use of the system input stream for PLAN input. The records will be 80 characters unblocked.

The second example shows a data set named MYFILE on a 2311, serial number MY2311, that will be used to honor PLINP calls with a NOD parameter of six. The records are 80 characters long and blocked five. (Note: If a PLINPxxx DD card is not found, a user ABEND code of 100 results.)

PLOUTxxx

This DD card defines the sequential PLAN output file written using the PLAN unit record subroutines. The xxx is a three-digit number equivalent to the NOD parameter in the subroutine call parameter list. The PLOUTxxx DD statement will be used as the initial output device. Only one PLOUTxxx DD statement is allowed per PLAN job step. The following examples show the PLOUTxxx DD statements:

```
//PLOUT100 DD SYSOUT=A

//PLOUT103 DD DSNAME=MYFILE,UNIT=2311,       X
//             VOLUME=SER=MY2311,            X
//             SPACE=(CYL,(10,5),            X
//             DISP=(,KEEP),DCB=(RECFM=FBA,X
//             LRECL=121,BLKSIZE=605)

//PLOUT107 DD UNIT=SYSCP
```

The first example shows the use of the system output stream for PLAN output. The record format assumed is 133 characters, unblocked, with the first character of the record used for carriage control. This device will be used whenever a PLOUT call is made with NOD variable = (100). The PLAN subroutine PCCTL can be used for carriage control.

The second example shows the use of a 2311 for PLAN output. The records are blocked five, and the data set MYFILE is to be retained at the end of the PLAN execution.

The third example shows the use of a card punch for PLAN output. The record size is 80 characters. (Note: If no PLOUTxxx DD card is found, PLAN will ABEND with the user code of 100.)

PLANLIB

This DD statement defines the PLAN library PDS, and should contain the load modules the user wishes to execute. In addition, this library must contain the modules DFJPSCAN, DFJRETN, and DFJPERRS, or PLAN execution is suppressed.

The following example shows a PLANLIB DD statement referring to a data set named

PLAN.MODLIB that resides on a 2311, serial number PLANPK:

```
//PLANLIB DD DSNAME=PLAN.MODLIB,DISP=OLD, X
//            VOLUME=SER=PLANPK,UNIT=2311
```

(<u>Note</u>: The PLANLIB DD statement may contain concatenated data sets.)

OPTIONAL DD CARDS

The following DD statements are optional and are used only if the job step requires them.

PLMANFIL

This DD statement defines a direct access data set that will be used to save and restore the managed COMMON array. PLAN uses this data set whenever a level 2, 3, or 4 command is processed, and the length of the managed array is not zero. This data set is required if the PLAN sort facility is used. If this DD card is not present and an attempt is made to either read or write the managed array a phrase abort condition occurs.

The following example shows the use of a temporary data set that will reside on any available direct access device. This data set will exist only while the PLAN job step is in execution.

```
//PLMANFIL DD UNIT=SYSDA,SPACE=(CYL,(2))
```

PLCHKPT

This DD statement defines a direct access data set that will be used to contain PLAN checkpoints. This data set is used by PLAN when a CALL LCHEX is executed. Note that in some cases the error routines (ERROR, ERRAT, ERREX, ERRET) do use LCHEX. If this data set is not present when a CALL LCHEX is executed, a phrase abort condition occurs.

The following example shows the use of a 2301 for checkpoint. The checkpoint file will be written using 5000-byte records.

```
//PLCHKPT DD UNIT=2301,VOLUME=SER=MY2301, X
//           SPACE=(CYL,(2)),             X
//           DCB=(BLKSIZE=5000)
```

PLANDRVx

This DD statement defines a direct access data set that will be used for the PLAN DYNAMIC files. The x is a single digit and refers to drives 0 through 7. This DD card is required if the user needs DYNAMIC PLAN files. PLAN DYNAMIC drive 0 is required if error message queueing is used. The following are examples of PLANDRVx DD statements:

```
//PLANDRV0 DD DSNAME=PLANDRVA,UNIT=2311,  X
//            VOLUME=SER=MY2311,DISP=OLD

//PLANDRV7 DD DSNAME=PLANDRVB,UNIT=2311,  X
//            VOLUME=SER=MY2311,          X
//            SPACE=(CYL,(10)),DISP=(,KEEP)
```

The first example shows the use of an existing data set named PLANDRVA as DYNAMIC drive 0. This data set must previously have been used as a PLAN DYNAMIC drive.

The second example shows the use of a data set named PLANDRVB as PLAN DYNAMIC drive 7. The data set will be formatted by the PLAN system and will be retained after the PLAN execution.

PLFSynnn

This DD statement defines the direct access data set associated with GDATA, RDATA, and WDATA call statements. Y is a single digit equivalent to the NDR parameter in the GDATA call. The nnn is a three-digit number equivalent to the file number in ID(1). This DD card is required if the user needs RWDATA type files. The following examples show PLFSYnnn DD statements:

```
//PLFS0007 DD DSNAME=PERMFIL,UNIT=2311,   X
//            VOLUME=SER=MY2311,DISP=OLD

//PLFS7043 DD DSNAME=NEWFILE,UNIT=2311,   X
//            VOLUME=SER=MY2311,          X
//            SPACE=(TRK,(20))8           X
//            DCB=(BLKSIZE=1024)
```

The first example shows the use of an existing data set named PERMFIL. This data set will be accessed on the GDATA call when ID(1) = 7 and NDR = 0.

The second example shows the use of a new data set named NEWFILE. This data set will be formatted with 1024-byte records. This data set will be accessed on the GDATA call when ID(1) = 43 and NDR = 7.

PLSEQxxx

This DD statement defines a sequential PLAN input/output data set to be read or written using the PLAN unit record subroutines. The xxx is a three-digit number equivalent to the NOD parameter in the subroutine call parameter list. The user may have as many PLSEQxxx cards as required. The following examples show PLSEQxxx DD statements:

```
//PLSEQ005 DD UNIT=(2400-2),LABEL=(,NL),   X
//             VOLUME=SET=TAPE,DISP=OLD,    X
//             DSNAME=INPUT.MASTER,         X
//             DCB=(TRTCH=ET,DEN=2,         X
//             LRECL=80,BLKSIZE=400,        X
//             RECFM=FB)

//PLSEQ007 DD UNIT=SYSDA,SPACE=(CYL,(2)), X
//             DCB=(RECFM=F,BLKSIZE=512)
```

The first example shows the use of a seven-track tape for program input. It contains 80 character records with a blocking factor of five. Note the name INPUT.MASTER indicates this data set is to be used for input only. This data set will be referenced when a CALL PLINP(5) is executed.

The second example shows the use of direct access for intermediate work files. This data set could be used for both input and output. It would be referenced by either a CALL PLINP(7) or a CALL PLOUT(7).

DATA SET CONSIDERATION

PLINPxxx/PLOUTxxx/PLSEQxxx

The data sets defined by these DD statements are used for sequential input and output by the PLAN system as well as the user modules. The xxx in the DD name is a three-digit number equivalent to the NOD parameter in CALL PLINP and in CALL PLOUT.

The PLAN system module DFJPSCAN, which processes the PLAN commands, uses one of these data sets. In DFJPSCAN, the PLINP calls use a NOD parameter of (0) which specifies the current PLAN input device. The PLOUT calls use a NOD parameter of (100) that specifies the current PLAN output device. The user may vary these devices during execution by using the PLAN subroutine IOCS. The PLINPxxx DD statement is used as the initial PLAN input device, and PLOUTxxx is used as the initial output device. The xxx suffix cannot be duplicated on PLINP, PLOUT, or PLSEQ statements.

The default for PLINP and PLSEQ data sets is 80-character unblocked records. This is comparable to a DCB parameter of DCB=(LRECL=80,RECFM=F;BUFNO=2). The user may specify, in his DD card, any valid BSAM DCB parameter for input devices. This data set may reside on a card reader, magnetic tape, or direct access device.

The default for PLOUT data sets is 133 character unblocked records, with the first character being used for standard ASA carriage control characters and the remaining 132 characters used as data. This is equivalent to a DCB parameter of DCB=(LRECL=133,RECFM=FA,BUFNO=2). The user may specify any valid BSAM DCB parameter for an output device. This data set may reside on a printer, card punch, magnetic tape, or direct access device. (Note: The BUFNO parameter specifies the number of buffers to use for reading or writing the data sets.) The default value is two. The user may conserve some core by specifying BUFNO = 1. However, this will degrade performance. There is no significant performance improvement in specifying a value greater than two in the BUFNO subparameter. If using a card reader or punch, and stacker selection is used via the PCCTL subroutine, BUFNO = 1 must be specified in the DCB parameter.

If any data set named in a PLINP, PLOUT or PLSEQ DD card resides on a tape or direct access device, it may be used for both input and output. The PLAN subroutine 'PENDF' allows the user to reverse the status of a sequential data set. The PLAN system normally opens data sets of this type with a parameter of INOUT to allow both reading and writing on the data set. When using file-protected tapes, a special situation occurs. The operating system data management routines require that the file-protect ring be present on any tape that could be written on. To process a file-protected tape, the user must specify a DSNAME parameter of which the first five litters must be INPUT. Example:

```
DSNAME=INPUT
DSNAME=INPUT.MASTER
```

When a DSNAME of this type is found, the PLAN system will open this data set on the tape for input only.

SEQUENTIAL FILE SUPPORT

The following steps outline the manner in which certain special conditions are handled on the OS/360 version of the PLAN I/O subroutines (PLINP/PLOUT/PEOF/PCCTL).

Two subroutines are provided under OS PLAN that allow specification of page length and status switching (CLOSE) for PLINP/PLOUT data sets.

CALL PPAGL(NOD,N) is a subroutine used to specify the number of lines to be used as the page length for those data sets containing printed output. If N is 0, a default of 60 is used. The maximum value of N is 32,767.

A call to PPAGL sets the current line count to the page length specified. It also forces the next carriage control operation to be a skip to 1, unless overridden by an intervening call to PCCTL.

CALL PENDF(NOD) is a subroutine that may be used to close a sequential data set. If a data set is in output status, an EOF is written after the last record. Both PLINP and PLOUT data sets are repositioned to the beginning of this data set.

1. Maximum record size for any input/output record is 32,760 characters.
2. Records may be blocked within the limits of the specified device. Truncated records are accepted if the character count is a multiple of the logical record length.
3. A PLINP/PLOUT call to an invalid device (missing DD card) is ignored.
4. The DCB RECFM parameter must be F, FA, FB, or FBA.
5. If the device is a printer, the DCB RECFM parameter must be FA.

6. In order to effect carriage control, that is, for PCCTL to be functional, the DCB RECFM parameter must be FA or FBA.

7. The following items define PCCTL functions:

   a. If the device is a reader, PCCTL will control stacker selection. DCB=(RECFM=F, BUFNO=1) must be used.
   b. If the device is a punch, RECFM must be FA for PCCTL to control stacker selection.
   c. If RECFM is FA or FBA, PCCTL will cause the correct ASA control character to be inserted as the first character of the record.

8. The following items are specifications for the PEOF routine.

   a. Logical EOF is set when:
      (1) A "UREND" is read by CALL PLINP. The logical EOF will be reset by the next CALL PLINP to the data set.
      (2) The line count is zero for output data sets (CALL PLOUT) using RECFM FA or FBA.
   b. Physical EOF is set when:
      (1) EOF is read by a CALL PLINP,
      (2) a call PLINP is issued to a device not capable of input,
      (3) a CALL PLOUT is issued to a device not capable of output,
      (4) A CALL PLOUT is issued to a data set in input status (A CALL PLINP had previously been issued).
      (5) A CALL PLINP is issued to a data set in output status (A CALL PLOUT had previously been issued).

9. The following specifications pertain to the carriage tape simulation functions on an output device (CALL PCCTL):

   a. The maximum page length is 32,767 lines.
   b. Default page length is 60 lines.
   c. If RECFM is FA or FBA, a line count is maintained and an automatic eject (skip to carriage channel 1) is set when the line count reaches zero.
   d. The maintenance of the line count is suspended when a PCCTL CALL is issued for a skip to channels 2-12.
   e. Maintenance of the line count is resumed when a CALL PCCTL is issued for a skip to channel 1.

A PLAN utility program (DFJPLENG) allows the user to set the page length to be used on an output file that is to contain data to be printed. This utility must be invoked by the standard PLAN command.

```
SET PAGE LENGTH, NOD xxx, PGL yyyyy;
```

where xxx is a number up to three digits equivalent to the NOD argument for the subroutines PLINP and PLOUT, and yyyyy is a number up to five digits to be used as the page length for the specified NOD.

PLSYSTAB

This data set is the PLAN system phrase dictionary and work data set. Its format is fixed-length records of 512 characters. PLSYSTAB must contain at least 14 records. The maximum record requirement is 268 records. If the data set defined in this card has a disposition of OLD, it must have been previously used for a PLAN phrase dictionary. This data set must reside on a direct access device.

PLMANFIL

This data set is used as a save area for the managed COMMON array whenever the commands being processed by PLAN are level 2, 3, or 4 and the length of the managed area is not 0. The size of this data set is dependent on the number of command levels the PLAN job is using and the length of the managed array. This data set is also used as working storage by the DYNAMIC file and PERMANENT file sort facility. The size of this file affects the performances of the sort. The sort facility requires space for at least one logical record; however, any additional space will be used to optimize the sort.

The default block size for this data set is 512 characters. The user may specify a DCB BLKSIZE parameter of any value up to the limits of the device on which the data set resides. If disposition is NEW, this data set is formatted by PLAN. If disposition is OLD, this data set must have been created by QSAM or BSAM. This data set must reside on a direct access device.

PLCHKPT

This data set is used by PLAN to write and read PLAN checkpoints. The size of this data set is dependent on the size of the programs being executed and the number of levels of checkpoints to be taken by the job.

The default block size for this data set is 512 characters. The user may specify a DCB BLKSIZE paramater of any value up to the limits of the device on which the data set resides. If disposition is NEW, this data set is formatted by PLAN. If disposition is OLD, the data set must have been created by QSAM or BSAM. This data set must reside on a direct access device.

PLANDRVx

The data sets defined by these DD statements are PLAN DYNAMIC drives. The x is a single digit number from 0 to 7 that denotes the drive number. The size of the data set depends on the number and size of the DYNAMIC files that will be processed on the DYNAMIC drive. These data sets have a fixed format of 600-character records. This may not be varied by the user. The minimum size for these data sets is 20 records. These data sets must reside on a direct access device. If disposition is NEW, PLAN will format these data sets. If disposition is OLD, these data sets must have been used previously as a PLAN DYNAMIC drive.

DYNAMIC FILE SUPPORT (OS PLAN)

The NALLO parameter provided with CALL FIND is used to optimize space allocation. The basic unit of allocation for an OS PLAN file is 1350 FORTRAN words.

Each logical file can contain up to 147 discontiguous allocations. Thus, if normal allocation is allowed as the file is written, the maximum file size is restricted to 220,500 FORTRAN words. If the NALLO parameter of the CALL FIND subroutine is utilized, the maximum file size is 49,150, 350 FORTRAN words.

Each DYNAMIC drive may contain a maximum of 149 discontiguous free areas. This means that in cases of extreme discontiguous allocation, a file may be destroyed.

To approximate the physical space to be allocated for the PLAN DYNAMIC drive, the following formula may be used:

```
  NT=((NW+1349)/1350*10*NF+10)/RT

  where: NT = number of tracks
         NW = average file length in words
         NF = number of files
         RT = number of 600-byte records
              per track
```

PLFSynnn

The data sets defined by these DD statements are those generated outside of PLAN by BSAM or QSAM. They are processed with the PLAN subroutines GDATA, RDATA, and WDATA. They must reside on a direct access device. Y in the DD name is a one-digit number from 0 to 7 and is equivalent to the NDR parameter in the GDATA call arguments. The nnn is a three-digit number from 1 to 255 equivalent to the value in ID(1). If disposition is OLD, the existing data set specifications will be used. If disposition is NEW, the user may specify the RECFM, LRECL, and the BLKSIZE DCB subparameter. The default block size for these data sets is 512 characters.

PERMANENT FILE SUPPORT

The OS version of PLAN provides support for files established outside of PLAN with the following characteristics:

1. File contains fixed-length records.

2. File may be organized as a sequential or direct access file.

3. No secondary allocation is provided.

4. Track overflow feature may not be used.

5. No keys are allowed.

6. There may be no control characters.

7. The file may contain no truncated records.

The logical drive number (NDR) and the logical file number (ID(1)) must be equivalenced to the data set name. The DDNAME "PLFSynnn" will establish a name/number equivalence between PLFSynnn and NDR/ID(1), where y corresponds to NDR and may range from 0-7, and nnn corresponds to ID(1) and may range from 1-255.

USE OF FORMATTED DATA SETS

The data sets named in the PLMANFIL, PLFSynnn, PLCHKPT, PLANDRVx, and PLSYSTAB DD statements must be formatted for successful PLAN execution.

The format for PLSYSTAB and PLANDRVx is fixed, and these data sets must be formatted by the PLAN initialization routine. This is done when a DISP(NEW,XXXX) is found in the DD statements. If DISP=NEW is specified on either of these data sets and the PLAN system issues the message DFJ999 *E* PLAN EXECUTION INHIBITED or the job terminates abnormally before PLAN initialization is complete, these data sets may not be formatted correctly for subsequent use with a DISP=OLD parameter. In this situation the user should SCRATCH the data set and rerun the job with DISP=NEW.

The formats for data sets named in the PLMANFIL, PLCHKPT, and PLFSYnnn DD statements are flexible. Any record size is allowable up to the device limits. The data sets will be formatted by the PLAN initialization routine if DISP=NEW is specified. They may, however, be formatted by any program using the BSAM or QSAM access method.

Since data set formatting requires that the entire data set be written, a significant reduction in the time required for PLAN initialization can be obtained by using preformatted data sets (DISP=OLD) for these DD names.

RULES FOR WRITING MODULES IN LANGUAGES OTHER THAN FORTRAN

Other languages may be used to generate modules suitable for loading and execution under the PLAN system provided they adhere to the following conventions:

LINKAGE REGISTERS

REGISTER NUMBER	NAME	FUNCTION
0	PARAMETER Return	Return answer value for function subroutines.
13	SAVEAREA Register	Address of the area in the calling program where the called program may store the contents of the general registers.
14	RETURN Register	The address in the calling program to which control is to be returned after completion of the called program.
15	ENTRY Register	Address of the entry point of the called program. This register may also be used by the called routine to return condition codes.
1	ARGUMENT LIST Reg.	Address of the argument list passed to the called program.

No other general register may be used to pass any parameter to or from a called program.

ARGUMENT LISTS

Each entry in an argument list must be four bytes long and must be aligned on a full word boundary. The first byte of each entry should contain zero. The last three bytes should contain the address of an argument. The first byte of the last entry in the argument list should have the high-order bit set to '1'.

SAVE AREAS

Any module invoked as a LOCAL under PLAN is presented with a standard 18 word save area. If a module issues any loader subroutine calls, the contents of this save area may be changed with the following exceptions:

Word 4. The contents of GPR14 (RETURN) on entry to the called program.

Word 7. The contents of GPR1 (address of the argument list) on entry to the called program.

Although the contents of the save area presented to a LOCAL module may be changed, the PLAN system ensures that, on return from a LOCAL module, the registers are restored correctly before returning to the calling program.

BLANK COMMON DECLARATION

If a blank COMMON control section is present in a program, it must be at least 640 32-bit words long.

For those languages that cannot generate a blank COMMON control section, the name PLANBCOM will be accepted as an alias for blank COMMON. Modules that use this alias may not have an actual control section named PLANBCOM.

LANGUAGE EXAMPLES

The following examples illustrating the rules defined previously show the same problem written in FORTRAN, COBOL, PL/I, and Assembly Language. The problem is to determine the volume of a box given the three dimensions. The dimensions, the volume, and a title read from a card are to be printed. If one or more of the dimensions is missing, an error message is to be printed.

The phrase used for this problem is:

```
ALTER PHRASE:  SAMPLE LANGUAGE PROBLEM,
I(1)DIMENSION-,-,-,            (4)FORTRAN-*
F'FORSAM', (5)COBOL-*F'COBSAM', (6)PLI-
*F'PLISAM', (7)ASM-*F'ASMSAM';
```

The phrase puts the dimensions as integers in the PLAN communication array positions 1, 2, and 3. These locations are initialized to the value FALSE so that the modules may ensure that all dimensions have been specified. The logical variables FORTRAN, COBOL, PLI, and ASM are used to determine which of the modules are to be used.

```
ALTER PHRASE:  SAMPLE LANGUAGE PROBLEM,I(1)DIMENSION-,-,-,
 (4)FORTRAN-*F'FORSAM',(5)COBOL-*F'COBSAM',(6)PLI-*F'PLISAM',
(7)ASMB-*F'ASMSAM';
PLAN JOB;     LON;
 SAM LAN PRO, DIM1,2,3,FORTRAN;
*******************FORTRAN OK************************DIMENSIONS         1       2       3
                                                     VOLUME             6

SAM LAN PRO, DIM 1,2,FORTRAN;
*******************FORTRAN DIM(3) UNDEFINED**********DIMENSIONS         1       2 *UNDEF*
SAM LAN PRO, DIM 1,2,3       ,COB;
*******************COBOL OK**************************DIMENSIONS         1       2       3
                                                     VOLUME             6

SAM LAN PRO, DIM 1,,3,COB;
*******************COBOL DIM (2) UNDEFINED***********DIMENSIONS         1 *UNDEF*       3
SAM LAN PRO, DIM 1,2,3,PLI;
*******************PL/I OK***************************DIMENSIONS         1       2       3
                                                     VOLUME             6

SAM LAN PRO, DIM(2)2,3,PLI;
*******************PL/I DIM (1) UNDEFINED************DIMENSIONS   *UNDEF*       2       3
SAM LAN PRO,DIM1,2,3,ASMB;
*******************ASSMBLY LANG OK*******************DIMENSIONS         1       2       3
                                                     VOLUME             6

SAM LAN PRO,DIM1,2,ASMB;
*******************ASSEMBLY LANG DIM(3) UNDEFINED****DIMENSIONS         1       2 *UNDEF*
SAM LAN PRO, DIM2,3,4,FORTRAN,COBOL,PLI,ASM;
****************** ASM ******************************DIMENSIONS         2       3       4
                                                     VOLUME            24
****************** PL/I *****************************DIMENSIONS         2       3       4
                                                     VOLUME            24
****************** COBOL ****************************DIMENSIONS         2       3       4
                                                     VOLUME            24
****************** FORTRAN ************************* DIMENSIONS         2       3       4
                                                     VOLUME            24
```

FORTRAN

The following FORTRAN module is written as a subroutine. This eliminates the FORTRAN error handling subroutines and allows PLAN to intercept program check errors. PLAN can then abort the phrase where required rather than aborting the whole job step.

```
      SUBROUTINE FORSAE
      INTEGER DIM (3)
      DIMENSION HDGS (8)
      COMMON L(625),LS(15),M(510)
      EQUIVALENCE,(DIM(1),M(1))
      DATA HDGS /'DIME','NSIO','NS','VOLU','ME',' ','*UND','EF*'/
C                 READ HEADING FROM PLAN INPUT DEVICE
      CALL PLINP (0)
C                 TRANSFER HEADING TO PLAN OUTPUT LINE
      CALL PBFTR (0,100)
C                 MOVE 'DIMENSIONS' TO PLAN OUTPUT LINE
      CALL PAOUT (100,54,12,HDGS(1))
      NERR = 0
      DO 10 I=1,3
C                 CHECK DIMENSION
      IF (NDEF(DIM(I))) 1,1,2
C                 MOVE '*UNDEF*' TO PLAN OUTPUT LINE,
C                        IF A DIMENSION IS NOT SPECIFIED
1     CALL PAOUT (100,59+I*8,7,HDGS(7))
      NERR = NERR + 1
      GO TO 10
C                 MOVE DIMENSION TO PLAN OUTPUT LINE
2     CALL PIOUT (100,59+I*8,7,DIM(I))
10    CONTINUE
C                 PRINT THE LINE
      CALL PLOUT (100)
C                 RETURN TO PLAN, IF ANY ERRORS HAVE BEEN FOUND
      IF (NERR) 11,11,99
C                 PRINT THE VOLUME
11    CALL PAOUT (100,54,12,HDGS(4))
      CALL PIOUT (100,67,7,DIM(1)*DIM(2)*DIM(3))
      CALL PLOUT (100)
C                 RETURN TO PLAN
99    CALL LRET
      GO TO 99
      END
```

COBOL

The following COBOL module illustrates three points which must be observed in using this language. Examples of these points are indicated in the module listing.

1. Locating PLAN COMMON. The module is written as a subroutine with one parameter. PLAN will pass it to the address of the switch words in COMMON. An ENTRY card must be supplied to the link editor with the name used in the COBOL ENTRY statement (ENTRY COBSAME for this example).

2. Parameters passed to PLAN subroutines. Integer parameters must occupy a 32-bit word, that is, PICTURE S9(9) COMPUTATIONAL.

3. Function subroutines. Since COBOL does not support function subroutines, they must be CALLed with an additional parameter. This parameter will receive the functional value. It is a 32-bit integer for the function NDEF, and a 32-bit floating-point number (COMPUTATIONAL-1) for PEOF, PCOMP, and PIOC. The absolute value returned by the function may vary from one execution to the next. It will be negative, zero, or positive, depending on the condition being tested in the function. Example: CALL 'PCOMP' USING A, B, N, PVAL.

   PVAL is the extra parameter used for returning the functional value. It will be negative if A is less than B, zero if A is equal to B, and positive if A is greater than B. These conditions correspond to the statement numbers 1, 2, and 3 in the Program Description Manual explanation of the function. Note also that an array must be in a contiguous core area. Example:

   ```
   02 C OCCURS 10 TIMES
   03 A COMPUTATIONAL-1.
   03 D COMPUTATIONAL-1.
   ```

A cannot be used as an array to be passed to PLAN.

```
IDENTIFICATION DIVISION.
PROGRAM-ID. 'COBSAM'.
DATA DIVISION.
WORKING STORAGE SECTION.
01  FILLER COMPUTATIONAL.
    02  NID                          PICTURE S9(9)  VALUE 0.
    02  NOD                          PICTURE S9(9)  VALUE 100.
    02  HDG-PP                       PICTURE S9(9)  VALUE 54.
    02  HDG-FW                       PICUTRE S9(9)  VALUE 12.
    02  DATA-PP                      PICTURE S9(9)  VALUE 67.
    02  DATA-FW                      PICTURE S9(9)  VALUE 7.
    02  I                            PICTURE S9(9)  .
    02  NERR                         PICTURE S9(9)  .
    02  VOLUME                       PICTURE S9(9)  .
    02  J                            PICTURE S9(9)  .
01  HDGS.
    02  DIM-HDG PICTURE X(12) VALUE 'DIMENSIONS'.
    02  VOL-HDG PICTURE X(12) VALUE 'VOLUME'.
    02  UND-HDG PICTURE X(08) VALUE '*UNDEF*'.
LINKAGE SECTION.
01  COMN COMPUTATIONAL.
①→  02  LS  OCCURS  15 TIMES PICTURE S9(9).
    02  DIM                          PICTURE S9(9)  OCCURS 3 TIMES.
PROCEDURE DIVISION.
    ENTER LINKAGE.
①→  ENTRY 'COBSAME' USING COMN.
    ENTER COBOL.
             NOTE READ HEADING FROM PLAN INPUT DEVICE.
    ENTER LINKAGE.
    CALL 'PLINP' USING NID.
    ENTER COBOL.

             NOTE TRANSFER HEADING TO PLAN OUTPUT LINE.
    ENTER LINKAGE.
    CALL 'PBFTR' USING NID, NOD.
    ENTER COBOL.

             NOTE MOVE 'DIMENSIONS' TO PLAN OUTPUT LINE.
    ENTER LINKAGE.
②→  CALL 'PAOUT' USING NOD, HDG-PP, HDG-FW, DIM-HDG.
    ENTER COBOL.
             NOTE CHECK AND PRINT DIMENSIONS.
    MOVE ZERO TO NERR.
    PERFORM DIM-CHECK VARYING I FROM 1 BY 1 UNTIL
        I IS GREATER THAN 3.

             NOTE PRINT THE FIRST LINE.
    ENTER LINKAGE.
    CALL 'PLOUT' USING NOD.
    ENTER COBOL.

             NOTE GO TO RETURN TO PLAN, IF ANY DIMENSION
               ERRORS HAVE BEEN DETECTED.
    IF NERR IS GREATER THAN ZERO GO TO CALL-LRET.

             NOTE PRINT THE VOLUME.
    COMPUTE VOLUME = DIM (1) * DIM (2) * DIM (3).
    ENTER LINKAGE.
    CALL 'PAOUT' USING NOD, HDG-PP, HDG-FW, VOL-HDG.
    ENTER COBOL.
    SUBTRACT 24 FROM DATA-PP.
    ENTER LINKAGE.
    CALL 'PIOUT' USING NOD, DATA-PP, DATA-FW, VOLUME.
    ENTER COBOL.
    ENTER LINKAGE.
    CALL 'PLOUT' USING NOD.
    ENTER COBOL.
             NOTE RETURN TO PLAN.
```

```
  CALL-LRET.
      ENTER LINKAGE.
      CALL 'LRET'.
      ENTER COBOL.


               NOTE CHECK DIMENSION.
  DIM-CHECK SECTION.
      ENTER LINKAGE.
3 --> CALL 'NDEF' USING DIM (I), J.
      ENTER COBOL.
      IF J IS POSITIVE GO TO DIM-OK.


               NOTE MOVE '*UNDEF*' TO PLAN OUTPUT, IF A
                 DIMENSION IS NOT SPECIFIED.
  DIM-ERR.
      ENTER LINKAGE.
      CALL 'PAOUT' USING NOD, DATA-PP, DATA-FW, UND-HDG.
      ENTER COBOL.
      ADD 1 TO NERR.
      GO TO DIM-UPDATE.


               NOTE MOVE DIMENSION TO PLAN OUTPUT.
  DIM-OK.
      ENTER LINKAGE.
      CALL 'PIOUT' USING NOD, DATA-PP, DATA-FW, DIM (I).
      ENTER COBOL.
  DIM-UPDATE.
      ADD 8 TO DATA-PP.
```

PL/I

The following PL/I module illustrates the points which must be observed in using PL/I. Examples of the first four points are indicated in the module listing.

1. Environment. The outside procedure must have OPTIONS (MAIN) specified. This permits the module to properly initialize the pseudo register vector. This means that program checks are handled by the PL/I error routine which may cause the job step to be aborted. ON statements may be used to abort the phrase and return to PLAN, if desired. Note that a program check in some PLAN subroutines may cause unpredictable results, since these subroutines use the register (register 12) which PL/I assumes is pointing to the pseudo register vector. The register is, of course, restored before returning to the PL/I module.

2. Parameters passed to PLAN subroutines. Integer and floating-point parameters must be 32-bit words (FIXED BINARY (31), FLOAT DECIMAL (6), or FLOAT BINARY (21)). They must be aligned on a full word boundary. Character and bit strings, arrays, and structures cause a dope vector to be created. This dope vector describes the data and specifies its location. PL/I passes the address of the dope vector instead of the address of the data. In the case of character strings it is necessary to declare a based variable whose pointer contains the address of the character string. In the example, HDG_ARRAY is based on HDG_ADDR which points to HDGS. Then HDG_ARRAY(1) is used as a parameter when DIM_HDG is intended. Note that in FORTRAN HDG_ARRAY and HDG_ARRAY(1) are equivalent when used as a parameter; in PL/I they are not. The latter form must be used when passing arrays to PLAN subroutines.

3. Function Subroutines. NDEF returns FIXED BINARY (31). PECF, PIOC, PCOMP return FLOAT DECIMAL (6) or FLOAT BINARY (21) (32-bit floating-point form).

   The absolute value returned may vary from one execution to the next. It will be negative, zero, or positive depending on the condition being tested, that is,

IF NDEF(ARG)<0 THEN ARG IS FALSE
IF NDEF (ARG)=0 THEN ARG IS TRUE
IF NDEF (ARG)>0 THEN ARG IS REAL (Not TRUE or FALSE)

These correspond to statement numbers 1, 2, and 3 in the Program Description Manual explanation of NDEF.

4. Locating PLAN COMMON. The assembler subroutine PLCOM may be used to set a pointer variable to point to the switch words in PLAN COMMON. In the example module, COMN is a structure based on COMN_ADDR which is set by PLCOM. COMN contains the 15 switch words, LS, as 32-bit integers, and the communication array.

   In the example, only the first three words of the communication array are used to supply the dimensions of the box. They are in the form of 32-bit integers.

5. Link-editing. The PL/I module must be fully link-edited to put the pseudo register vector together properly and determine its length which must be known upon entry to the PL/I module. Therefore, the PLAN loader link edit feature cannot be used for PL/I LOCAL's.

6. PL/I Multitasking. Only the following PLAN subroutines are re-entrant:

   NDEF
   PARGI, PARGO
   PBTST
   PCOMP
   PHTOE
   PPACK, PUNPK, BREAK
   STVAL, GTVAL
   TRUE, FALSE

```
PLISAM: PROCEDURE OPTIONS (MAIN);<----(1)

     DCL (PAOUT, PIOUT) ENTRY (FIXED BINARY (31), FIXED BINARY (31)<----(2)
         FIXED BINARY (31), FIXED BINARY (31));
     DCL 1 COMN BASED (COMN_ADDR),
         2 LS (15) FIXED BINARY (31), <-----------(4)
         2 DIM (3) FIXED BINARY (31);
     DCL 1 HDGS STATIC,
         2 DIM_HDG CHAR (12) INITIAL ('DIMENSIONS'),
         2 VOL_HDG CHAR (12) INITIAL ('VOLUME'),
         2 UND_HDG CHAR (8) INITIAL ('*UNDEF*'),
         2 NID FIXED BINARY (31) INITIAL (0),
         2 NOD FIXED BINARY (31) INITIAL (100);
     DCL HDG_ARRAY (8) BASED (HDG_ADDR) FIXED BINARY (31);
     DCL NDEF RETURNS (FIXED BINARY(31));


                  /*  LOCATE COMMON*/
     CALL PLCOM (COMN_ADDR); <----------(4)

                  /* READ HEADING AND TRANSFER TO PLAN OUTPUT LINE */
     CALL PLINP (NID); CALL PBFTR (NID,NOD);
     HDG_ADDR = ADDR (HDGS);

                  /* MOVE 'DIMENSIONS' TO PLAN OUTPUT LINE*/
     CALL PAOUT (NOD, 54,12,HDG_ARRAY(1));
                  /* HDG_ARRAY(1) MUST BE USED INSTEAD OF DIM HDG */
                  /* SINCE PL/I DOES NOT PASS THE ADDRESS OF      */
  (2)             /* CHARACTER STRINGS. INSTEAD, IT PASSES THE    */
                  /* ADDRESS OF A DOPE VECTOR WHICH DESCRIBES     */
                  /* THE LINE.                                    */
     NERR=0;
DIM_CHECK: DO I=1 TO 3;

                  /* CHECK THE DIMENSION  */
     IF NDEF(DIM(I)) > 0 THEN GO TO DIM_OK;<----(3)

                  /* MOVE '*UNDEF*' TO THE PLAN OUTPUT LINE,      */
                  /* IF THE DIMENSION IS UNDEFINED.               */
DIM_ERR: CALL PAOUT (NOD,59+I*8,7,HDG_ARRAY(7));
     NERR=NERR+1;
     GO TO DIM_UPDATE;
```

```
                  /* MOVE THE DIMENSION TO THE PLAN OUTPUT LINE      */
DIM_OK: CALL PIOUT (NOD,59+I*8,7,DIM(I));
DIM_UPDATE:END;

                  /* PRINT THE FIRST LINE                            */

     CALL PLOUT (NOD);

                  /* GO TO RETURN TO PLAN, IF ANY OF THE             */
                  /* DIMENSIONS ARE UNDEFINED                        */
     IF NERR > 0 THEN GO TO CALL_LRET;

                  /* PRINT THE VOLUME                                */
     CALL PAOUT (NOD,54,12,HDG_ARRAY(4));
     CALL PIOUT (NOD,67,7,DIM(1)*DIM(2)*DIM(3));
     CALL PLOUT (NOD);

                  /* RETURN TO PLAN                                  */
CALL_LRET: CALL LRET;
     END PLISAM;

         COM
LDR      DS    625F
LS       DS    15F
PLCOM    CSECT                ←——— 4
         USING *,15
         L     1,0(1)
         MVC   0(4,1),=A(LS)
         BR    14
         END
```

ASSEMBLY LANGUAGE

The following listing illustrates methods of interfacing to the PLAN system and locating and using BLANK COMMON in assembly language.

On entry to every module loaded by PLAN, the GP registers are set as follows:

15 Entry point of the module
14 Return address in the PLAN loader which will simulate a CALL LRET
13 Address of a standard 18 word save area
12 Address of BLANK COMMON
1 Address of a parameter list. If the module is not a PLAN LOCAL or no parameters are passed by the calling module, a standard parameter pointing to the switch words is passed.

There are three methods of locating COMMON in assembly language modules.

1. The address of a parameter list pointing to the switch words is passed in GPR1.

2. The address of COMMON is passed in GPR12.

3. A COMMON control section COM operation code may be declared in the assembly which describes BLANK COMMON.

In cases 1 and 2 a DSECT may be described and the registers used for direct addressing. This is shown in the listing. In case 3, adcons referencing the names described in the COM control section may be used to reference any or all the items in COMMON. For example:

```
         L     REG12,=V(COMMON)
         USING COMMON,REG12
         •
         •
         •
         COM


COMMON   DS    625F
LS       DS    15
DIM1     DS    F
DIM2     DS    F
DIM3     DS    F
```

The second point illustrated by the example module is the difference in linkage and save area conventions. In the example module, the registers are not saved nor is a save area provided. This is allowable since the save area passed by PLAN may be used. The subroutine LRET will return control to PLAN correctly. Modules written using the standard OS conventions will function correctly.

```
         PRINT ON,NOGEN
*  ASSEMBLY LANGUAGE MODULE FOR PLAN SYSTEM
ASMSAM   CSECT
DOCNTR   EQU   3
SUBSCRPT EQU   4
LINPTR   EQU   5
DIMPTR   EQU   6
NERR     EQU   7
         LR    11,15                   SET OUR BASE
         USING ASMSAM,11
         USING COMMON,12
*  PLAN PASSES THE ADDRESS OF COMMON IN GPR12
*  IT IS NOT NECESSARY TO SAVE REGISTERS OR ESTABLISH A SAVE AREA
         CALL  PLINP,F0,VL             READ A CARD
         CALL  PBFTR,(F0,F100),VL      TRANSFER TO OUTPUT BUFFER
         CALL  PAOUT,(F100,F54,F12,HDGS1),VL PUT "DIMENSION" IN LINE
         XR    NERR,NERR               RESET ERROR INDICATOR
*  CHECK FOR PRESENCE OF ALL 3 FACTORS
         LA    DOCNTR,3                3 TIMES THRU
         XR    SUBSCRIPT,SUBSCRIPT     SET SUBSCRIPT TO 1
         LA    LINPTR,67               INITIALIZE POSITION POINTER
         LA    DIMPTR,DIM1             POINT AT ARG IN COMMON
DOLOOP   ST    LINPTR,LINPOS           SET LINE POSITION
         LA    DIMPTR,DIM1(SUBSCRPT)   LOCATE AND SET
         ST    DIMPTR,NDEFARG          ARGUMENT ADDRESS
         OI    NDEFARG,X'80'           INDICATE LAST ARG
         LA    1,NDEFARG               POINT AT IT
         CALL  NDEF                    AND TEST IT
         LTR   0,0                     CHECK ANSWER REG
         BH    DIMOUT                  BR IF ARG OK
         CALL  PAOUT,(F100,LINPOS,F7,HDGS7),VL PRINT 'UNDEF'
         BCT   NERR,CONTINUE           SET ERROR INDICATOR
DIMOUT   LA    1,DIMOUTL               POINT AT ARG LIST
         CALL  PIOUT                   AND OUTPUT FACTOR
CONTINUE LA    SUBSCRPT,4(0,SUBSCRPT)  STEP TO NEXT ARG
         LA    LINPTR,8(0,LINPTR)      STEP LINE POSITION
         BCT   DOCNTR,DOLOOP           TEST AND DECR LOOP COUNTER
         CALL  PLOUT,F100,VL           PRINT LINE
         LTR   NERR,NERR               ANY ERROR
         BM    RETURN                  BR IF YES
*  CALCULATE VOLUME AND PRINT IT
         LM    0,1,DIM1                GET FIRST TWO FACTOR
         MR    0,0                     ONE * TWO
         M     0,0                     * THREE
         ORG   *-2
         DC    S(DIM3)
         ST    1;VOLUME                SET ANSWER
         CALL  PAOUT,(F100,F54,F12,HDGS4),VL PUT 'VOLUME' IN
         CALL  PIOUT,(F100,F67,F7,VOLUME),VL OUTPUT VALUE
         CALL  PLOUT,F100,VL           PRINT LINE
*  RETURN TO PLAN
RETURN   CALL  LRET
F0       DC    F'0'
F7       DC    F'7'
F12      DC    F'12'
F54      DC    F'54'
F67      DC    F'67'
F100     DC    F'100'
HDGS1    DC    CL12'DIMENSIONS'
HDGS4    DC    CL12'VOLUME'
HDGS7    DC    CL7'*UNDEF*'
*  VARIABLES
VOLUME   DC    A(0)
LINPOS   DC    A(0)
*  PARAMETER LISTS
DIMOUTL  DC    A(F100)
         DC    A(LINPOS)
         DC    A(F7)
```

```
NDEFARG  DC    A(0)
*  DESCRIPTION OF COMMON AREA
COMMON   DSECT
LOADER   DS    625F
         DS    15F                     SWITCH WORDS
DIM1     DS    1F                      MANAGED ARRAY (1)
DIM2     DS    1F                      MANAGED ARRAY (2)
DIM3     DS    1F                      MANAGED ARRAY (3)
         END
```

PLAN CORE MANAGEMENT

Once the PLAN system is initiated it maintains control over the entire region or partition.

The PLAN system divides a partition into five major areas:

1. PLAN BLANK COMMON

2. PLAN PROGRAM AREA

3. NON-MANAGED OS FREE STORAGE

4. MANAGED FREE STORAGE

5. PLAN SYSTEM AREA

The user should be aware of the function of these areas and the manner in which PLAN controls each.

Figure 4 illustrates the PLAN system allocation of main storage within a partition or region.

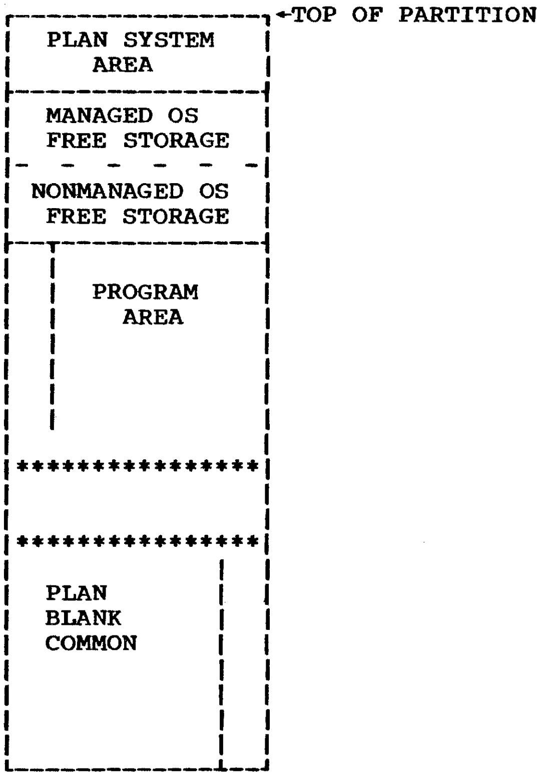


Figure 4. Main storage allocation

PLAN BLANK COMMON AREA

PLAN BLANK COMMON always resides at the beginning of the partition or region. It is variable in length but must be at least 640 words long. PLAN BLANK COMMON is used as a communication area by all program modules loaded by the PLAN system.

When loading modules, the PLAN loader deletes the BLANK COMMON control section from the module and relocates all program references to BLANK COMMON to point to the PLAN BLANK COMMON area. The first 2560 bytes (640 FORTRAN words) of this area are reserved for PLAN system use and must not be altered by any user program. Therefore, the COMMON statement in every program must specify a dummy array of 640 words to ensure protection of this area. For example:

```
    COMMON L(640), J(10), K(20),...
```

Any alteration of that part of PLAN BLANK COMMON reserved for PLAN system use will probably cause abnormal termination of the PLAN job step.

The length of the PLAN BLANK COMMON area may be altered whenever a new module is loaded into the program area. It will be as long as required by any resident load module but never shorter than the length specified as a data variable in loader Switch Word 9. (See Problem Language Analyzer (PLAN) Program Description Manual (H20-0594) for an exact description of the PLAN Switch Words.) Regardless of the requirements of currently resident modules and the contents of Switch Word 9, the length of PLAN COMMON may not be less than 640 words (2560 bytes).

PROGRAM AREA

The PLAN PROGRAM area is located above the BLANK COMMON area. The BLANK COMMON area and PROGRAM area are one contiguous core area extending from the bottom of the partition to some variable point in the partition.

For a program to be loaded, core is allocated from the top of the PROGRAM area towards the BLANK COMMON area. This means that as programs are loaded the end of the PROGRAM area extends itself towards the top of BLANK COMMON. The area between the top of BLANK COMMON and the end of the PROGRAM area is always considered available for program loading and although addressable by the user, the contents of this area is not protected by the PLAN system.

OS FREE STORAGE AREA

Programs executed under OS/360 have the ability to request dynamic allocation and deallocation of core areas outside the absolute program area. This facility is provided in OS by use of the GETMAIN and FREEMAIN macros. Programs loaded by the PLAN loader must have the same facility available to them. The free storage area in a PLAN partition is used to honor GETMAIN requests from problem programs.

There are two requirements in this area:

1. Requests for temporary space to be used only by the requesting module

2. Requests for permanent space that can be used to pass arrays, data sets, etc., across load module boundaries

For this reason PLAN splits the free storage area into two sections (1) the managed area, and (2) the nonmanaged area. These two areas are treated as individual subpools of free storage and PLAN facilities are provided to set the length of each of these areas and dynamically switch between using either of them.

The PLAN system maintains several pointers concerned with the MANAGED FREE STORAGE area. Whenever a program segment is released, the PLAN system uses these pointers to perform the following maintenance:

1. DELETE modules that the segment loaded via the LOAD macro.

2. Close data sets that were left open by the segment.

3. Use the FREEMAIN macro to release all core obtained by the segment's use of the GETMAIN macro.

Management of the core resources require that supervisor state coding be used for MVT to release subpools 251 and 252.

The subroutines DFJUMC and DFJUNC also employ supervisor state code and all three systems to optimize the blanking and unblanking of the MANAGED AND NONMANAGED FREE STORAGE area.

The supervisor state is entered through use of an SIO appendage routine which alters the current PSW for the PLAN job.

The user must be aware of the implications of the above maintenance procedures. Programs that reside in lower-level (higher-segments) that are called as LOCALs may issue the GETMAIN macro only for temporary use. Whenever a segment is released, all areas in MANAGED FREE STORAGE obtained by the GETMAIN macro are released. This includes both the segment and all modules or subroutines called as LOCALs by the segment.

The NONMANAGED FREE STORAGE area is declared by the NFS operand of the PARM keyword in the EXEC job control card.

If a NONMANAGED FREE STORAGE area is declared, it is the user's responsibility to maintain this area.

Two subroutines are provided to allow the user to control the area of OS FREE STORAGE that is used to honor GETMAIN requests.

CALL DFJUMC sets the system status so that the managed area of OS FREE STORAGE is used for GETMAINs.

CALL DFJUNC sets the system status so that the nonmanaged area of OS FREE STORAGE is used for GETMAINs.

PLAN SYSTEM AREA

This area of the partition is reserved at initialization time for PLAN system use. It contains the control blocks and arrays, and PLAN system subprograms that will be required for the entire execution of the PLAN system.

PLAN INITIALIZATION

The following discusses the partition/region initialization under PCP-MFT/MVT. When PLAN is initially entered, the partition/region is as shown below:

```
      PCP-MFT              MVT
   ┌──────────┐       ┌──────────┐
   │TIOT      │       │          │ 2K SP252
   ├──────────┤       ├──────────┤
   │SAVE AREA │E      │SAVE AREA │
   ├──────────┤       ├──────────┤
   │          │       │          │
   │          │       │          │ 2K SP0
   │          │       ├──────────┤
   │          │       │          │
   │          │       │          │
   │- - - - - │C      │- - - - - │
   │          │       │          │
   │          │       │          │
   │          │       │          │
   │          │       │          │
   ├──────────┤B      ├──────────┤B
   │          │       │          │
   │PLAN      │       │PLAN      │
   │MAINLINE  │       │MAINLINE  │12K SP251
   │          │A      │          │A
   └──────────┘       └──────────┘
```

LEGEND
  N = E-B
  M = B+N-C

The PLAN PROGRAM/COMMON area must be defined contiguously. Therefore, the core allocation for this area must be defined first. This is accomplished by issuing a GETMAIN macro for all of memory followed by a FREEMAIN macro of all memory not required for PLAN/COMMON.

Upon entry to PLAN, the issuance of a GETMAIN VC for between 8 bytes and 16 million bytes will return the address B (see preceding chart) and the length N. The address C (end of program area) can be calculated as 1024*L+A where L is the value specified for the PGAR parameter in the EXEC DFJPLAN control card. With the value for C and M, a FREEMAIN macro is issued and allocation of the program area is complete.

The PLAN system area is then created. The required PLAN modules are loaded via the LOAD MACRO. In PCP-MFT, these modules are loaded at the highest possible core address. In MVT, a new 4K block of SP251 is created above and adjacent to the program area. The data sets are opened next. Access methods, I/O areas, DCBS, etc., are allocated from the top of free storage. Additional allocations to SP252 may be required for access methods on MVT.

If the NFS parameter has been specified, a NONMANAGED free storage area is allocated using a technique similar to that used to allocate the PROGRAM/COMMON area. Therefore, after PLAN initialization memory is as shown in the following diagram:

```
        PCP-MFT                  MVT
   ┌──────────────┐          ┌─────────┐
   │              │          │ACCESS   │
   │TIOT          │          │METHODS  │ 2K SP252
   ├──────────────┤          ├─────────┤
   │SAVE AREA     │          │SAVE AREA│
   ├──────────────┤          ├─────────┤
   │PLAN MODULES  │          │I/O AREA │2K SP0
   ├──────────────┤          ├─────────┤
   │ACCESS METHODS│          │ACCESS   │
   │I/O AREAS     │B         │METHODS  │ 2K SP252
   ├──────────────┤          ├─────────┤
   │              │         B│I/O AREAS│2K SP0
   │              │          ├─────────┤
   │              │          │         │
   │              │          │         │
   │              │         A│         │
   │              │          ├─────────┤
   │              │          │PLAN     │
   │              │A         │MODULES  │ 4K SP251
   ├──────────────┤          ├─────────┤
   │              │          │         │
   │PROGRAM       │          │PROGRAM  │
   │AREA          │          │AREA     │nK SP0
   ├──────────────┤          ├─────────┤
   │              │          │         │
   │PLAN          │          │PLAN     │
   │MAINLINE      │          │MAINLINE │12K SP251
   └──────────────┘          └─────────┘
```

N=B-A

PROGRAM LOADER

Since the BLANK COMMON control section must be deleted from all load modules and references to this CSECT relocated to PLAN COMMON, the PLAN system cannot use the OS/360 FETCH facility. In effect, the PLAN program loader replaces the OS/360 FETCH facility.

The following restrictions apply to modules that are loaded th the PLAN loader:

1. Use of XCTL is prohibited in PLAN modules. The use of LINK is allowed.

Any program that is "linked" to by a module loaded by the PLAN loader may use the XCTL, LINK or ATTACH macros but may not use any PLAN subroutine that includes a blank COMMON specification. The linked-to program may also be in overlay mode.

2. The "overlay structure" is not supported in PLAN modules, except as defined in 1.

3. Load modules may not be in overlay or scatter mode or contain TESTRAN symbol cards.

4. Load modules must be marked as executable by the link editor.

In addition to loading programs, the PLAN program loader provides the user with the following capabilities:

1. Load time link editing

2. Access to the RAM, LINKPAC, and JOBPAC areas without use of the LOAD macro

3. Automatic management of the program area which eliminates the need for using the LINK, LOAD and DELETE macros

4. Use of an in-core PDS directory for frequently loaded modules to improve loader performance

Figure 5 is a simplified logic diagram of the PLAN program loader.

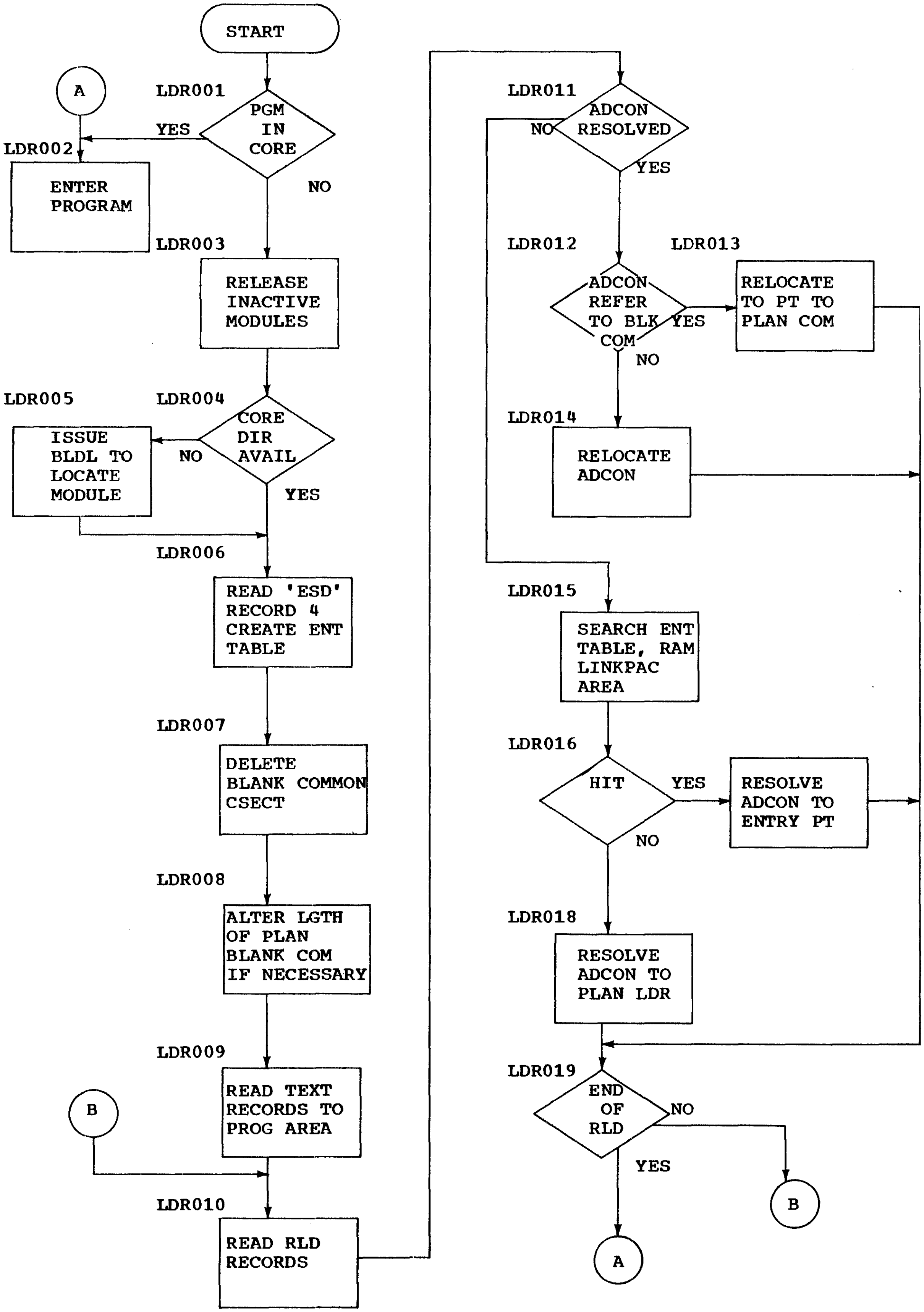


Figure 5. Program Loader

The first step in loading a module is to determine if it is already in the program area. If it is, the program is entered without any further processing. If the module is not in the program area, inactive modules currently in the program area are released and the space occupied by them reclaimed. This procedure is transparent to the user and keeps the maximum amount of space available for program loading.

The next step in the loading process is to locate the load module in the PLAN library data set. If an in-core directory is available, it is searched for the module name. If the name is not found, a BLDL macro is issued to locate the load module. After locating the load module the ESD records (External-Entry Symbol Table) are read. From these records, an entry-point table is built for the module being loaded. This table may be used to resolve ADCONS when subsequent modules are loaded. Then the BLANK COMMON CSECT is located and deleted from the module and the length of PLAN BLANK COMMON is altered if necessary. The TXT records which contain the relocatable code for the module are then read into the program area. The RLD records (Relocatable Adcon Dictionary) are read and the adcons in the load modules are relocated. If an adcon refers to BLANK COMMON it is relocated to point to PLAN BLANK COMMON. All ADCONS referencing points within the load module are relocated. If an unresolved external reference (V-type ADCON) is found, all entrypoint tables for modules already in core, the JOBPAC area and finally the LINKPAC or RAM area are searched. If an equivalent entry point is found in any of these, the external reference is resolved to this entry point. This gives direct access to the JOBPAC and LINKPAC areas to FORTRAN programs without the need for programming assembly language linkage subroutines. If the external reference cannot be resolved to an entry point in core, it is resolved to point to the PLAN loader in such a way that an execution time reference to the ADCON causes the named program to be loaded and entered as a PLAN LOCAL.

EXECUTION-TIME LINKAGE EDITING

Because the PLAN loader has full control of the region or partition, it can resolve references between load modules that were not link-edited together before execution.

While loading a module, all unresolved ADCONS pointing to entries in in-core segments will be resolved. ADCONS that cannot be resolved directly are resolved indirectly through the PLAN loader, which will treat a reference to an unresolved ADCON as a CALL LOCAL.

Unresolved branch type (v) ADCONS that are resolved to the PLAN loader are restricted in that execution time references to the ADCON must be direct for example:

```
    L    15,=V(NAME)
    BALR 14,15
```

Offset referencing as shown below will not function correctly and will probably cause termination of the PLAN JOB step. In other words, IBCOM= cannot be called as a LOCAL.

```
    L   15,=V(NAME)
    BAL 14,N(0,15)
```

The two sets of coding shown below are equivalent and correct. The V-CON for SUBRTN in set 2 may be unresolved following link-editing.

```
SET1
  REAL*4 NAME(2)/'SUBRTN'/
    •
    •
    •
  CALL LOCAL(2,NAME,ARG1,ARG2,ARG3)
    •
    •
    •
  END
SET2
    •
    •
    •
  CALL SUBRTN (ARG1, ARG2, ARG3)
    •
    •
    •
  END
```

USE OF THE LINKPAC AND RAM AREAS

A PLAN utility program (DFJLLIST), that gives the PLAN system the capability of referencing the LINKPAC or RAM area, is provided. This utility must be invoked by the PLAN command:

```
    CREATE LOADER ENTRIES:  (NAME1,...);
```

where NAME1,... is a load module name that is to be loaded into the partition via the LOAD macro and be made available as entry points for the execution of any loader call. This allows programs in the LINKPAC or RAM areas to be objects of a CALL LOCAL. The names specified in the LIST must be in the JOBLIB PDS. To add this phrase to the dictionary, the following PLAN command must be executed:

```
    ADD PHRASE:  CREATE LOADER ENTRIES, PRO
    'DFJLLIST';
```

The maximum number of names in the list is 75. Use of this command destroys any entries defined by previous use of the command.

Programs that reference blank COMMON may not be operands of this command.

USE OF IN-CORE DIRECTORY

A PLAN utility program (DFJCRDIR) allows the user to build an in-core PDS directory of names of frequently loaded modules. This utility must be invoked by the PLAN command:

```
    CREATE CORE DIRECTORY:  (NAME1,...);
```

NAME1,... is a load module name that is placed in the in-core PDS directory to decrease load time for those modules. The names in the list must be entries in the PLANLIB PDS.

Use of this command will replace the previous directory. The maximum number of entries is 75 names.

This facility is added to the PLAN language dictionary (PFILE) by executing the following command:

```
    ADD PHRASE:  CREATE  CORE  DIRECTORY,
    PROGRAM 'DFJCRDIR';
```

OVERLAY PROCESSING

Although the PLAN program loader does not allow program modules to be in overlay mode, the PLAN system provides a flexible overlay processing capability.

In the simplest forms of processing, programs may succeed one another in the program area as their names are found in the pop-up list. Many applications, however, require that functionally dependent programs reside in core concurrently. This implies that preplanning must go into developing a TREE STRUCTURE of overlays.

One of the principle features of the PLAN system is that load modules sharing core do not have to be link-edited together. Because of this, overlay processing under PLAN is possible without preplanning an entire overlay tree structure, and in fact, the user may dynamically alter his tree structure at execution time on the basis of the amount of main storage and other system resources available.

The PLAN subroutines LOCAL and LEX provide the user with a means of constructing and executing a tree structure of almost any complexity.

The LOCAL subroutine function is similar to the OS LINK macro. If a copy of a module is already available in the program area, it is used. If a copy is not available, a new program segment is loaded and is subordinate to the caller.

The LEX subroutine performs a function equivalent to the OS XCTL macro in that the calling module may be overlayed by the called module.

In order to use an overlay processing technique under PLAN the user should be familiar with the controls that the PLAN system exercises over problem programs.

Several terms are defined below that are used in describing PLAN program execution control.

A SEGMENT is one or more modules brought into the program area by a single program load request. A segment is loaded only when a request is made for a module that is not in the program area. When parenthetical grouping is used in the pop-up list, a segment may consist of more than one module; the first module named in the group is considered the initial entrypoint for the segment. As each segment is loaded, it is assigned a hierarchial level number. The level used in the maintenance of the program area.

A LOCAL is defined as any module or program invoked by the PLAN LOCAL facility. This facility is used whenever the LOCAL subroutine is called or an unresolved external ADCON is referenced in a problem program.

EXECUTION LEVEL is defined as the depth of subprograms that have been executed (the CALL LOCALs that have been executed without associated returns).

The following defines how PLAN controls the loading of segments and management of the program area.

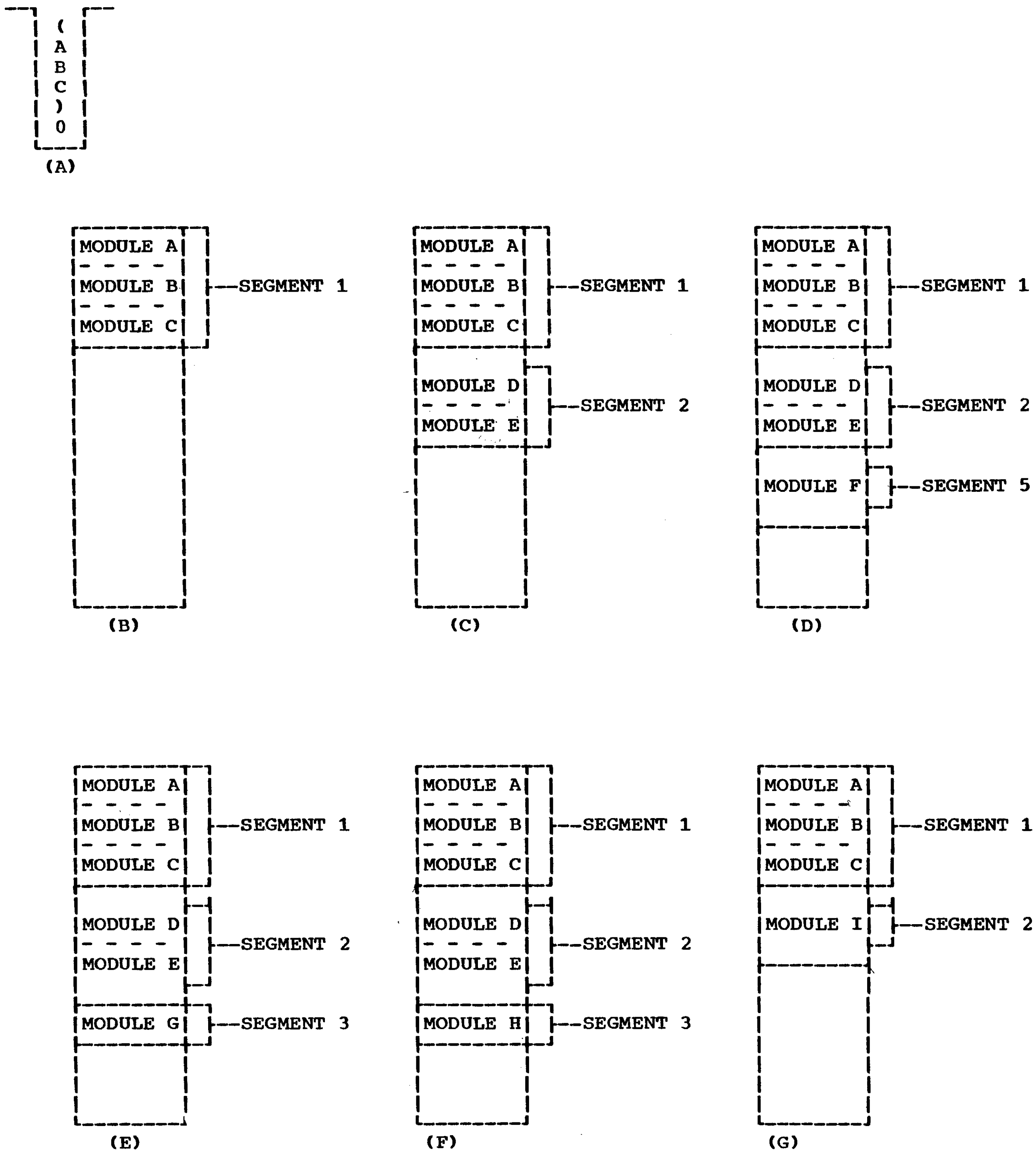


Figure 6. Pop-up list

Assume that upon entry to the PLAN execution monitor, the pop-up list is as shown in Figure 6A. The list defines three program modules named A, B, and C that are to be loaded as one logical segment and defines that program A is to be entered and executed.

The program loader is called. Upon return the program area appears as shown in Figure 6B.

Program A is entered. Execution level is set to one. Program A issues a CALL LOCAL transferring the names D and E in parenthesis to the pop-up list.

Program D is not in core so the loader inspects the program area to see if any inactive program segments can be released. All program segments assigned a level less than or equal to the current execution level must be retained when a CALL LOCAL is issued. Segment one in the program area is retained because execution level is one.

PLAN loads programs D and E as a segment and assigns it a level of two. The program area is as shown in Figure 6C. Program D is entered and the execution level is incremented because of the CALL LOCAL and is now two.

Program D now issues a CALL LOCAL to program E. Since E is already in core, a program load is not required and E is entered. The execution level is incremented to three because of the CALL LOCAL. Program E calls LOCAL to program C.

Program C is already in core in segment one so it is entered. The execution level is set to four.

Program C calls LOCAL to program F. F is not in core so a segment must be loaded. The loader determines if any program segments can be released. Execution level is at four. Therefore, segment level one and two must be retained. Program F is loaded as a segment and assigned a level of five. The program area is as shown in Figure 6D. The level assigned to a segment on a CALL LOCAL is always one higher than the current execution level. This prevents releasing a segment which may be active in the local chain. In sequences of CALL LOCALs issued requesting residual modules, the execution level is incremented but no loading is required. Adjacent segments are not necessarily assigned sequential level numbers (shown by the level assigned to program F).

Program F is entered. The execution level is set to five. F returns to C. The execution level is decremented to four. C returns to E. The execution level is set to three. E returns to D. The execution level is set to two.

Program D now issues a CALL LOCAL to G. Program G is not in core. The loader determines if any segments can be released. Execution level is at two. Therefore, all segments assigned a level higher than this can be released. Program F (in segment five) is released.

Program G is loaded and assigned a level of three. The program area appears as shown in Figure 6E. Note that program G has overlayed program F. The execution level is set to three. G is entered. Program G issues a CALL LEX to program H.

On a CALL LEX, all segments that are assigned a level equal to or higher than the current execution level must be released. In this case, segment three (contains the calling program) is released.

Program H is loaded and is assigned a level equal to the current execution level. The execution level remains the same because of the CALL LEX. The program area is as shown in Figure 6F. Program H returns to D. The execution level is decremented to two.

Program D returns to A. The execution level is set to one. Program A issues a CALL LOCAL to program I. I is not in core so the loader releases inactive segments. Execution level is at one so segments two and three can be released.

Program I is loaded and is assigned a segment level of two. The program area is as shown in Figure 6G. I is entered and the execution level is set to two. Program I issues a CALL LOCAL to program B.

Program B is in core in segment one and is entered. The execution level is set to three. B returns to I. The execution level is set to two. I returns to A. The execution level is set to one.

Program A now zeros the pop-up list and returns. Since a return was executed from execution level one and since the pop-up list is zero, the PLAN system will initiate processing of the next command.

The preceding description of the PLAN loader is not intended as a practical example of using an overlay structure under PLAN. The description does illustrate that the LOCAL and LEX subroutines facilities provide the FORTRAN programmer with a dynamic control of the program area.

The significant differences between the LOCAL and LEX subroutines facilities and the OS LINK and XCTL macros are:

1. Communication between the caller and the called modules may be through blank COMMON as well as by parameter list.

2. Called programs may use subroutines or modules in the calling segment without passing external names as arguments.

3. Program modules may be used recursively.

4. A LOCAL may be cancelled by a higher priority program by using LNRET.

The bank loading facility (parenthetical grouping of names in the pop-up list) allows the user to include, in any segment, commonly used programs used as subprograms by the structure developed below the segment. This conserves core and loading time. This facility also allows the user to optimize the use of the program area at execution time based on its length.

The user is provided with special arguments that, when encountered in the pop-up list, indicate the limits of the functionally dependent modules. The left parenthesis indicates the start of a string of module names for which the user desires coexistent residence. The right parenthesis indicates the end of the string. Figure 7 represents the pop-up list containing a list of programs. Programs M0716 through M0725 are to be grouped in memory concurrently.

```
-----┐     ┌----
     |M0712|
     |M0756|
     | (   |
     |M0716|
     |M0796|
     |M0732|
     |M0725|
     | )   |
     |M0749|
     | 0   |
     └-----┘
```

Figure 7. Loader pop-up list

The systems programmer in determining the scheduling control, that is, which modules may coexist within the partition, must recognize and/or account for the following conditions:

1. If more modules are grouped (bounded in the pop-up list with parentheses) than can coexist, those modules that will not fit are not loaded concurrently.

2. If space can be found, all parenthetically grouped modules are loaded into the partition with the entry to the program named following the left parenthesis.

3. Loading of a module results only if the module does not already exist in memory.

4. If the left/right parenthesis is encountered when entering data into the pop-up list without a corresponding right/left parenthesis, the unmatched parenthesis is ignored. Therefore, parenthetically grouped programs must be added to the pop-up list with a single loader subroutine call.

5. If the left or right parenthesis is to be inserted in the pop-up list, it must be left-justified in two FORTRAN words

6. Program lists, verb lists, and check-entry program lists include the parenthetical groupings in literal form. Example:

   ...,PROGRAMS 'M0713, (M0726, M0733, M0792), M0796',...

7. The combination of the parenthetical program grouping and the use of command input of program lists gives the user the power to add segments (modules) to his root structure at execution time.

8. If all programs indicated in the coexistent grouping cannot be loaded because of insufficient partition size, the right parenthesis is floated forward in the pop-up list to include those programs for which coexistent loading was accomplished.

   The original right parenthesis is deleted and a right parenthesis is regenerated in the pop-up list at a position that indicates the last program which was successfully loaded.

9. A call with a negative value of N is required to interrogate the pop-up list for successful loading of the coexistent programs.

10. Parenthetical grouping is acceptable but ignored on the 1130 version of PLAN.

11. The left and right parentheses and all programs associated with the indicated coexistent grouping must be added to the pop-up list with a single call to the PLAN loader subroutines, or both parentheses must be included in a PHRASE-defined program list.

12. All program lists to be inserted into or to be extracted from the pop-up list must begin on a full-word boundary.

RETURN LINKAGE

The FORTRAN RETURN statement functions exactly like the CALL LRET PLAN loader call. Register 14 is used to cause a return from the mainline (logic module) to the PLAN loader. PLAN modules that contain CALL LNRET or that are reentered at a primary entry may not exit via RETURN. FORTRAN subroutines which modify variables passed to them as arguments must use the FORTRAN return statement.

CALL EXIT should be used to terminate a module to assure that buffers have been purged and data sets closed when FORTRAN (non-PLAN) I/O is incorporated within a module.

PARAMETER PASSING

If the arguments in a parameter list are external names, the called program and calling program must be compiled by the same level FORTRAN compiler.

PLAN SYSTEM CHECKPOINT

The following regulations govern execution and control of the checkpoint facility within the OS version of PLAN (CALL LCHEX):

1. Checkpoints can be reloaded only within the limits of the phrase from which they were written. This means that any checkpoint that has not been reloaded when the end of the phrase is encountered -- that is, when the pop-up list is found to be empty -- is destroyed. No warning message is issued.

2. If the checkpoint return (*) is encountered while in local mode, the local processing is terminated and the checkpoint is reloaded.

3. Any input/output error while reading or writing the checkpoint data set results in a phrase abort and PLAN level error recovery is initiated. This action is also true when insufficient space is available in the checkpoint data set.

4. The user may specify, in the DCB BLOCKSIZE parameter of the PLCHKPT DD card, the record size (in bytes) to be used when writing checkpoints. If no blocksize is specified, a blocksize of 512 is assumed.

5. There is no logical restriction on the number or level of checkpoints that a user may execute. A physical limit based on the size of the checkpoint data set may produce a real limit or error condition as outlined in 2 above.

6. Checkpoint restarts are executed in a reverse order from which they are written, that is, last in-first out.

7. After a checkpoint is taken, the status of all data sets, except system data sets (those data sets processed by CALL PLINP, CALL PLOUT, CALL GDATA, and CALL FIND), must not be altered until the checkpoint is restarted. This is a user responsibility and no check is made by PLAN to prevent such an alteration. If a data set status is altered while a checkpoint is in effect, the results are unpredictable.

8. COMMON is not protected between the time that a checkpoint is taken and the restart is loaded. It is the user responsibility to save and reload those parts of COMMON that might be destroyed and that must be present for continued execution of the checkpointed module.

9. Floating-point registers are not restored when a checkpoint is restarted.

USER-EXIT PROGRAMMING

The DFJPSCAN user-exit programs must be written to expect the standard /360 FORTRAN subroutine linkage conventions.

IOCS DEVICE PARAMETERS

Under System/360 OS PLAN, INPUT and LIST correspond to units defined as DD names defined in the JCL for the PLAN job. The value specified for INPUT or LIST, corresponds to the device specified as the PLAN input device PLINPnnn in the job description deck. Unit nnn specified for LIST, corresponds to the device specified as the PLAN output device PLOUTnnn.

PROGRAMMING RESTRICTIONS

The following System/360 FORTRAN statement should not be used because of its detrimental effect on the execution of PLAN. Alternate facilities are listed for the statement. To avoid overriding the PLAN processor or endangering another user's job, should not be executed.

CALL DUMP   This statement creates a premature end to the PLAN execution. Therefore, the CALL PDUMP, followed by a CALL LRET should be used.

PERMANENT FILE SORT/MERGE

CALL GSORT(ID) and CALL GMERG(ID,JD,KD) provide the identical function for PERMANENT files as provided by CALL PSORT and CALL PMERG do for DYNAMIC files.

ESTIMATING STORAGE REQUIREMENTS

In order to determine the size of the partition or region required to run a PLAN job, the user must know (1) the length of the PROGRAM/COMMON area, (2) the amount of storage required to honor GETMAINs, and (3) the amount of storage required for the PLAN system area.

The length of the PROGRAM/COMMON and OS FREE STORAGE areas is largely determined by the size of the program modules to be loaded by the PLAN system and the amount of storage obtained by GETMAINs in the problem programs.

The PROGRAM/COMMON area must be at least 19,500 bytes long to accommodate the PLAN module DFJPSCAN, (the system interpreter). If the user employs DFJPSCAN user exits, additional storage equal to the length of the user-exit modules is required.

Once in execution, the PLAN system issues GETMAINs in two areas as follows:

1. The program loader requires 16 bytes for every entry point in a module being loaded and 16 bytes for every unresolved ADCON in the module.

2. The error processing subroutines require 72 bytes for every call where a checkpoint is required.

The PLAN system uses the FREEMAIN macro to release all storage obtained when the storage is no longer needed. A minimum OS/FREE STORAGE area of 2048 bytes is suggested.

The length of the PLAN system area varies, depending on the PARM options selected in the EXEC statement and the number of DD cards defined. The table below shows the PLAN system area storage requirements.

FUNCTION	LENGTH IN BYTES
PLAN TABLES	552
PLAN SYSTEM MODULES	
DFJLODER	3200
TRACE OPTIONS	
DFJTRACE	424
DATA SET REQUIREMENTS	
PLANLIB	264
PLINPxxx	136+(# BUFFERS*(BLKSIZE+84))
PLOUTxxx	136+(# BUFFERS*(BLKSIZE+84))
PLSEQxxx	136+(# BUFFERS*(BLKSIZE+84))
PLMANFIL	128+(# BUFFERS*(BLKSIZE+168))
PLCHKPT	128+(# BUFFERS*(BLKSIZE+168))
PLFSynnn	128+(# BUFFERS*(BLKSIZE+168))
PLSYSTAB	128+(# BUFFERS*(680))
PLANDRVx	128+(# BUFFERS*(768))
OS/360 ACCESS METHODS (ESTIMATE)	
IGG019AV	88
IGG019BA	384
IGG019BB	104
IGG019BC	248
IGG019CC	80
IGG019CE	128
IGG019CF	240
IGG019CH	128
IGG019CI	136
IGG019CK	96
IGG019CL	80
IGG019KA	1360
IGG019KE	288
IGG019KK	176
IGG019KU	456
IGG019LI	232

The access method subroutines and the PLAN modules DFJLODER and DFJTRACE are reenterable, and may be placed in the RAM or LINKPAC area.

If the user employs the PLAN utility phrases CREATE CORE DIRECTORY or CREATE LOADER ENTRIES, additional storage is required in the PLAN system area.

For a core directory, the storage requirement is (8-36N), where N is the number of names in the operand of the CREATE CORE DIRECTORY phrase.

For a loader entry list, the storage requirement is (8-(12N)+A), where N is the number of names in the CREATE LOADER ENTRY phrase, and A is the amount of storage required to load any of the named modules into the partitions that are not in the RAM or LINKPAC areas.

STANDARD PLAN COMMANDS

This section discusses the statements distributed as a standard part of the PLAN system. The only command that is a programmed portion of PLAN is ADD PHRASE. All other commands must be added to the system through use of ADD PHRASE. This section provides a discussion of the facility provided by a set of these phrases that are entered into the language definition dictionary (PFILE or DFJPFIL) as a part of the PLAN system generation. Spacing within the phrase definitions may not accurately represent that of distributed commands.

ADD PHRASE

This command is added to the language definition dictionary when it is initialized.

ADD PHRASE: ADD PHRAS, (1)0, LEVEL0, I(-13)1, PROGRAM 'DFJPHRAS';

ADD PHRASE may be altered to list all added phrases by adding DFJPIDMP to the program list.

DELETE PHRASE

DELETE PHRASE provides the ability to remove commands from the language definition dictionary.

ALTER PHRASE: DELETE PHRASE, (1)-1, LEVEL0, I(-13)1, PROGRAM'DFJPHRAS';

DELETE PHRASE may be altered to list all deleted commands by adding DFJPIDMP to the program list.

ALTER PHRASE

ALTER PHRASE provides the ability to delete an existing version of a phrase and replace it with a new copy.

ADD PHRASE: ALTER PHRASE, I(1)-1,LEVEL0, I(-13)1,PROGRAM 'DFJPHRAS,DFJPHRAS';

ALTER PHRASE may be altered to list all altered commands by adding DFJPIDMP to the program list.

PLAN JOB

ALTER PHRASE: PLAN JOB, LEVEL 0, I(-1) FILE, SAVED, TO, LISTS, LB, LC, LD, ERASE, COMMON, MANAGED, NERM, DEVICE, I(1)SHORT-, LONG-, STACK-, IMMEDIATE-, DRIVE0, DFI-, PFI-,(-11)UMOD, I(-13)FORM0, $0 FORM:(LONG) ?=FORM+1, FORM:(IMM)?=FORM+2 FORM:(DFI)?= FORM+4, FORM:(PFI)?=FORM+8, TO=TO+DRIVE *2048;

PLAN JOB FUNCTION	NAME	CAP	MODE	DEFAULT VALUES	CHECKING RULES	EXPRESSIONS
SAVED STATEMENT FILE	FILE	(-1)	I	**NOTE		
INITIAL SAVED STATEMENT	SAVED	(-2)	I			
END SAVED STATEMENT	TO	(-3)	I			=TO+DRI*2048
DATA LIST A POINTER	LISTS	(-4)	I			
DATA LIST B POINTER	LB	(-5)	I			
DATA LIST C POINTER	LC	(-6)	I			
DATA LIST D POINTER	LD	(-7)	I			
ERASABLE COMMON POINTER	ERASE	(-8)	I			
SIZE OF COMMON	COMMON	(-9)	I			
SIZE OF MANAGED ARRAY	MANAGED	(-10)	I			
ERROR FILE QUEUE COUNT	NERM	(-11)	I			
DIAGNOSTIC DEVICE	DEVICE	(-12)	I			
DIAGNOSTIC MODULE(*NOTE)	UMOD	(-11)	LI			
DIAGNOSTIC FORMAT	FORM	(-13)	I	0		:(LON)?=FORM+1 :(IMM)?=FORM+2 :(DFI)?=FORM+4 :(PFI)?=FORM+8
SHORT FORM INDICATOR	SHORT	(1)	LOG	FALSE		
LONG FORM INDICATOR	LONG	(2)	LOG	FALSE		
STACKED ERROR INDICATOR	STACK	(3)	LOG	FALSE		
IMMEDIATE ERROR IND.	IMM	(4)	LOG	FALSE		
SAVED STATEMENT DRIVE	DRIVE	(5)	I	0		
DYNAMIC FILE ERROR IND.	DFI	(6)	LOG	FALSE		
PERMANENT FILE ERROR INDICATOR	PFI	(7)	LOG	FALSE		

*NOTE: "UMOD" and "NERM" are mutually exclusive and may not be used together.

**NOTE: Default values are not provided because the 15 PLAN switch words are automatically reset as a result of the execution of any Level 0 command.

PLAN JOB provides initialization functions for any PLAN run. This command, or one that provides the functions of this command, should be the first command processed when PLAN is invoked. The command meets the requirement that a level 0 phrase be the first phrase processed and sets the parameters controlled by the system switch words. System accounting functions may be conveniently facilitated by adding the name of an accounting module as a program list to this command. A sample of the command at execution time is:

```
PLAN JOB, MANAGED 200, ERASABLE 240,
COMMON 900, LISTS 30,60,200,209, SAVED 20
TO 30 FILE 3, DRIVE 2 SHORT, STACKED,
DEVICE 102;
```

The above example illustrates:

1. The setting of the managed array to a size of 200 PLAN words.

2. The establishing of the beginning of erasable COMMON at CAP 240.

3. The defining of the total size of COMMON to 900 PLAN words.

4. The defining of four CAP indices (30,60, 200,209) used in referencing a maximum of four data lists.

5. The execution of statements 20 to 30 in file 3, drive 2.

6. The designating of the short form of diagnostic.

7. The specification of the indicator to cause error stacking (STACKED).

8. The designation of the device upon which error messages are to be printed (DEVICE 102).

The following parameter discussions (see table above) give a breakdown of the PLAN JOB options:

1. SAVED STATEMENT FILE. This parameter defines the DYNAMIC file number (1-255) from which a saved statement is to be executed as the next statement. The parameter will not be used if the next PLAN command is to be read from the standard PLAN input device.

2. INITIAL SAVED STATEMENT. If the next PLAN statement is to come from a saved statement file, this parameter defines the number of the first statement that will be executed. If this parameter is specified, the FILE, DRIVE, and TO parameters should also be specified.

3. END SAVED STATEMENT. If saved PLAN statements are to be executed next, this parameter defines the highest-numbered saved statement that will be executed.

4. DATA LIST POINTER. This parameter is used to define the CAP indices for up to the maximum of four possible data lists. These data lists may be referenced by PSCAN for storing data, by PARGO and PARGI for transmitting data, and by user program modules.

5. LB. This parameter provides a direct pointer to the second of the data lists defined above.

6. LC. This parameter provides a direct pointer to the third of the data lists defined above.

7. LD. This parameter provides a direct pointer to the fourth of the data lists defined above.

8. ERASABLE COMMON. This parameter defines the communication array subscript that is to be treated as the beginning of erasable COMMON. Erasable COMMON extends from the CAP position identified to the end of the communication array. This parameter must be set to some positive value within the range of the communication array in order for many of the standard PLAN commands to execute. This switch word is reset to 490 each time a level 0 command is encountered.

9. SIZE OF COMMON. This parameter defines the total size of COMMON (including communication array, switch words, and resident loader).

10. SIZE OF MANAGED ARRAY. This parameter defines the number of PLAN words that are to be managed according to the level structure of the commands to be processed. If this value is set to a positive integer and statements have a level assignment the managed array save file must be present for the saving of data.

11. ERROR FILE QUEUE COUNT. If error diagnostics are to be written onto logical file 255 of logical drive 0 instead of directly to an output device, then this parameter will specify the maximum number of messages that are to be allowed on the file before the messages are to be written to the diagnostic device.

12. DIAGNOSTIC MODULE. This parameter is used to specify the name of a user-written module that is to process error conditions rather than using the normal system processing. Note that this option precludes the error queue option and is in lieu of writing the diagnostics onto the diagnostic device.

13. DIAGNOSTIC DEVICE. If a diagnostic module is not specified, this parameter specifies the sequential file device code upon which the diagnostics are to be printed. This switch word is reset to 100 each time a Level 0 command is encountered.

14. DIAGNOSTIC FORMAT. This parameter should not be referenced by a user. It is set as a result of use of items 15, 16, 17, 18, 20 and 21.

15. SHORT. The word "SHORT" is specified if the short-form option is desired. Short-form diagnostics mean that the phrase being processed when the error is detected is not listed with the error.

16. LONG. This parameter is used to set the long-form diagnostic indicator.

Long-form diagnostics include the EBCDIC image of the phrase which caused the error, along with the diagnostic message.

17. STACK. This parameter sets the indicator to cause error stacking. In this mode of processing, errors are written to the output device only when the error module is scheduled by the PLAN loader or when the stack overflows. If the stack overflows, the checkpoint facility must be used to allow scheduling of the error module.

18. IMMEDIATE. This parameter sets the indicator to cause diagnostics to be written to the output device one-by-one as they are encountered. The checkpoint file and checkpoint programming must be available to function in the IMMEDIATE mode.

19. SAVED STATEMENT DRIVE. This parameter specifies the PLAN DYNAMIC Drive number that will be used when the SAVE statements are processed.

20. DYNAMIC FILE ERROR INDICATOR. This parameter determines the PLAN system error procedures when an error is detected by the DYNAMIC FILE support subroutines.

21. PERMANENT FILE ERROR INDICATOR. This parameter determines the PLAN system error procedures when an error is detected by the PERMANENT FILE support subroutines.

SET LITERAL

SET LITERAL is the command used to define standard literals for storage into a GDATA type file. The literals are maintained in a manner that makes them accessible to the subroutine PHIN.

```
SET LIT, NAME'PLITF', NUMBERn, 'LITERAL',
FILEj, DRIVEm;
```

```
ADD PHRASE: SET LITERAL, PROGRAM'DFJP-
DIAG', I(-8)M, I(M)FILE254, I(M+1) NAME0,
I(M+4)DRIVE0, I(M+5)NUMBER-*RA' UNDEFINED
LITERAL NUMBER', I(M+6)LITERAL0, (M+1)TEST-
*TA'UNDEFINED FILE NAME':
(NAME>0)&(NAME<9);
```

SET LITERAL FUNCTION	NAME	CAP	MODE	DEFAULT VALUES	CHECKING RULES	EXPRESSIONS
ERASABLE COMMON POINTER	M	-8	I			
LITERAL FILE NUMBER	FILE	M	I	254		
LITERAL FILE NAME	NAME	M+1	I	0		
LITERAL FILE DRIVE	DRIVE	M+4	I	0		
LITERAL NUMBER	NUMBER	M+5	I	FALSE	*RA	
LITERAL TEXT	LITERAL	M+6	I	0		
TEST FILE NAME	TEST	M+1		FALSE	*TA	:(NAME>0) & (NAME<9)

1. ERASABLE COMMON POINTER. This parameter points to the position within the communication array defined as erasable COMMON. This parameter (switch word 8) is normally set with the PLAN JOB command.

2. LITERAL FILE NUMBER. This parameter defines a number to be used to process the GDATA type literal file.

3. LITERAL FILE NAME. This parameter defines the name of the GDATA file in which literal processing occurs. Note that this parameter must be given. Otherwise, the check entry defined under "test file name" will fail and the phrase will not be executed.

4. LITERAL FILE DRIVE. This parameter defines the PERMANENT drive on which literal file is located. Failure to provide this parameter results in the

assumption that the file is on PERMANENT drive zero.

5. LITERAL NUMBER. This parameter defines the identification number for the literal to be processed. Note that failure to supply a literal number will result in a phrase abort error diagnostic.

6. LITERAL TEXT. This parameter provides the literal text for the literal to be added to the file. If this parameter is not provided (literal length zero), the existing literal of the same number is removed from the file.

7. TEST FILE NAME. (See "Literal File Name" above).

LIST LITERALS

LIST LITERALS is a command that produces a listing of all literals maintained in a specified literal file.

ADD PHRASE: LIST LITERALS, LEVEL 1, PROGRAM'DFJPLITL', I(1)254, NAME-*A'LITERAL FILE NAME NOT DEFINED', I(5)DRIVE0, NOD100, (35)"NUMBER LENGTH TEXT OF PLAN LITERAL";

LIST LITERAL FUNCTION	NAME	CAP	MODE	DEFAULT VALUES	CHECKING RULES	EXPRESSIONS
LITERAL FILE NUMBER	FILE	1	I	254		
LITERAL FILE NAME	NAME	2	I	FALSE	*A	
LITERAL FILE DRIVE	DRIVE	5	I	0		
LITERAL OUTPUT DEVICE	NOD	6	I	100		

1. LITERAL FILE NUMBER. This parameter defines a number to be used to process the GDATA type literal file.

2. LITERAL FILE NAME. This parameter defines the name of the GDATA file in which literal processing occurs. Note that this parameter must be given. Otherwise, the check-entry defined under "test file name" will fail and the phrase will not be executed.

3. LITERAL FILE DRIVE. This parameter defines the PERMANENT drive on which literal file is located. Failure to provide this parameter results in the assumption that the file is on PERMANENT drive zero.

4. LITERAL OUTPUT DEVICE. This parameter defines the output device that is to be used to list the literals. The standard parameter results in the use of the current PLAN output device.

COMMUNICATION ARRAY DUMPS

DUMP COMMON is a command that produces a hexadecimal printout of the communication array. Identical print lines are suppressed.

DUMP MANAGED is a command that produces a hexadecimal printout of the managed portion of the communication array. Identical print lines are suppressed.

DUMP NONMANAGED is a command that produces a hexadecimal printout of the nonmanaged portion of the communication array. Identical print lines are suppressed.

DUMP SWITCHES is a command that produces a hexadecimal printout of the PLAN switch words.

Note carefully that these are blank-level phrases. Any attempt to use them following a PLAN phrase abort error will result in the phrase being skipped.

ALTER PHRASE: DUMP SWITCHES, I(-8)M, I(M)NNN-2, (M+11)A"SWITCHES", "LENGTH", I(M+15), NOD100, PROGRAM'DFJPCDMP';

ALTER PHRASE: DUMP COMMON, I(-8)M, I(M)NNN0, 'MANAGED ARRAY', 'NONMANAGED ARRAY', "SWITCHES", "LENGTH", I(M+15)NOD100, PROGRAM'DFJPCDMP';

ALTER PHRASE: DUMP MANAGED, I(-8)M, I(M)NNN1, 'MANAGED ARRAY', "SWITCHES", "LENGTH", I(M+15)NOD100, PROGRAM 'DFJPCDMP';

ALTER PHRASE: DUMP NONMANAGED, I(-8)M, I(M)NNN-1, (M+6)B'NONMANAGED ARRAY', "SWITCHES", "LENGTH", I(M+14)NOD100, PROGRAM 'DFJPCDMP';

DUMP FUNCTION	NAME	CAP	MODE	DEFAULT VALUES	CHECKING RULES	EXPRESSIONS
ERASABLE COMMON DEFINITION	M	-8	I			
FUNCTION SWITCH	NNN	M	I			
DUMP COMMON				0		
DUMP SWITCHES				-2		
DUMP MANAGED				-1		
DUMP NONMANAGED				+1		
OUTPUT DEVICE	NOD	M+15	I	100		

1. ERASABLE COMMON DEFINITION. This parameter, a pointer to that portion of the communication array to be used as erasable COMMON, is normally set by the PLAN JOB command.

2. FUNCTION SWITCH. The appropriate value within the word (0, -2, 1, -1) distinguishes between DUMP, DUMP SWITCHES, DUMP MANAGED, and DUMP NONMANAGED functions, respectively.

3. OUTPUT DEVICE. This parameter defines the sequential device code to be used for output.

FILE DUMPS

ALTER PHRASE: DUMP DYNAMIC, I(-8)M, I(M) FILE255, I(M+2)START0, I(M+3)END0, I(M+4) DRIVE0, (M+5)A"DRIVE FILE LENGTH", (M+12) NAME' 'I(M+15)NOD100, 1, PROGRAM 'DFJPFDMP';

ALTER PHRASE: DUMP PERMANENT, I(-8)M, I(M) FILE 255, I(M+2)START0, I(M+3)END0, I(M+4) DRIVE0, (M+5)A"DRIVE FILE LENGTH", (M+12) NAME' ', I(M+15)NOD100, 0, PROGRAM'DFJPFDMP';

DUMP DYNAMIC is a command that produces a hexadecimal printout of the PLAN DYNAMIC file. Identical print lines are suppressed.

DUMP PERMANENT is a command that produces a hexadecimal printout of a PLAN PERMANENT file. Identical print lines are suppressed.

The limits of the dump are defined by the START and END operands. If these are omitted, the entire file is dumped.

Note carefully that these phrases are blank level, and will therefore be skipped if PLAN level recovery is invoked as a result of an error in a nonblank-level phrase.

DUMP PERMANENT FUNCTION	NAME	CAP	MODE	DEFAULT VALUES	CHECKING RULES	EXPRESSIONS
ERASABLE COMMON INDEX	M	-8	I			
FILE NUMBER	FILE	M	I	255		
START OF DUMP	START	M+2	I	0		
END OF DUMP	END	M+3	I	0		
DRIVE	DRIVE	M+4	I	0		
FILE NAME	NAME	M+12	LI	BLANK		
OUTPUT DEVICE	NOD	M+15	I	100		
DUMP TYPE SWITCH		M+16	I	0,1		

1. ERASABLE COMMON INDEX. This index defines the location within the communication array known as ERASABLE COMMON. The index is normally set by the PLAN JOB command.

2. FILE NUMBER. This parameter defines the file number of the file that is to be dumped.

3. START OF DUMP. This parameter defines the number of the PLAN word within the file at which the file dump is to start.

4. END OF DUMP. This parameter defines the number of the last PLAN word within the file that is to be dumped. If the parameter is not given (parameter is set to zero), the full length of the file will be dumped.

5. DRIVE. This parameter defines the PLAN DYNAMIC or PERMANENT drive number on which the file to be dumped is located.

6. FILE NAME. This parameter defines the name of the file to be dumped.

7. OUTPUT DEVICE. This parameter defines the sequential device code that will be used for output.

8. DUMP TYPE SWITCH. This parameter determines whether a DYNAMIC or a PERMANENT file is to be dumped.

STATEMENT SAVE COMMANDS

ALTER PHRASE: SAVE, I(-1)SW, -1, I(-8)M, I(M)FILE 0, I(M+1)DRI-1, $0 SW:(FIL>0)?=FIL, SW(3):(DRI>-1) & (DRI<5)?=DRI*2048;

SAVE is a command to allow saving of the PLAN statements that follow the SAVE command on a PLAN logical file. Each statement to be saved must be prefixed with a statement number. Saving of statements is terminated by (1) a SEND command, (2) any command that does not have a statement number, or (3) another SAVE command.

SAVE FUNCTION	NAME	CAP	MODE	DEFAULT VALUES	CHECKING RULES	EXPRESSIONS
	SW	-1	I			
		-2	I	-1		
ERASABLE COMMON POINTER	M	-8	I			
DYNAMIC FILE	FILE	M	I	0		
DYNAMIC DRIVE	DRIVE	M+1	I	-1	*NOTE	

*NOTE: $0SW:(FIL>0)?=FIL, SW(3):(DRI>-1)&(DRI<5)?=DRI*2048

1. DYNAMIC FILE. This parameter defines the number of the PLAN DYNAMIC file on which the following statements are to be saved. If the parameter is omitted, the current file number in Switch Word 1 will be used.

2. DYNAMIC DRIVE. This parameter defines the number of the DYNAMIC drive on which the following PLAN statements are to be saved. If this parameter is omitted, the current value in Switch Word 3 divided by 2048 will be used as the DYNAMIC drive indicator.

ALTER PHRASE: SEND;

SEND is a command used to terminate the saving of a series of PLAN statements.

ALTER PHRASE: EXECUTE, I(-1)SW,0, I(-8)M, I(M)FROM0, I(M+1)TO 0, I(M+2)FILE 0, I(M+3) DRIVE-1, (M)F*TA"INVALID STATEMENT NUMBER OR DRIVE; $0SW:(FIL>0)?=FIL, DRI:(DRI<0)?= SW(3)/2048-.5!:$5, DRI:(DRI<0)?=0, $5FRO¬((DRI>-1)&(DRI<5))?=+, SW(3):(TO>0)?=DRI* 2048+TO!=DRI*2048, SW(2):(FRO>0)?=FRO FRO:(SW(2)>0);

EXECUTE FUNCTION	NAME	CAP	MODE	DEFAULT VALUES	CHECKING RULES	EXPRESSIONS
	SW	-1	I			
	M	-8	I			
FIRST COMMAND TO EXECUTE	FROM	M	I	0	*NOTE 1	
LAST COMMAND EXECUTED	TO	M+1	I	0		
STATEMENT FILE NUMBER	FILE	M+2	I	0		
STATEMENT DRIVE NUMBER	M+3	I	-1			
PARAMETER CALCULATIONS						*NOTE 2

*NOTE 1: *TA'INVALID STATEMENT NUMBER OF DRIVE'

```
*NOTE 2: $0SW:(FIL>0)?=FIL,
         DRI:(DRI<0)?=SW(3)/2048-.5!:$5,
         DRI:(DRI<0)=0,
         $5FRO:¬((DRI>-1)&(DRI<5))?=+,
         SW(3):(TO>0)?=DRI*2048+TO!=DRI*2048,
         SW(2):(FRO>0)?=FRO,
         FRO:(SW(2)>0)
```

1. ERASABLE COMMON POINTER. This parameter defines the location within the communication array of ERASABLE COMMON. The pointer is normally set by the PLAN JOB command.

2. DYNAMIC DRIVE. This parameter defines the PLAN DYNAMIC drive number that is to be used to process SAVED statements. If this parameter is omitted, the current drive specified by Switch Word 3 divided by 2048 will be used.

3. DYNAMIC FILE. This parameter defines the PLAN DYNAMIC file number that is to be used to process SAVED statements. If this parameter is omitted, the current save file specified by Switch Word 1 will be used.

4. FIRST SAVED. This parameter defines the number of the first SAVED statement to be executed. If this statement cannot be located, an error message (PSTSV-DFJ172) will be produced.

5. LAST SAVED. This parameter defines the highest-numbered SAVED statement to be executed. Execution continues from the first SAVED statement identified through continually higher-numbered statements to the statement identified with this parameter. If this parameter is omitted, only the statement indicated by Switch Word 2 will be executed.

```
SAVE, FILE 2, DRIVE 3;
6  A;
9  B;
18 C;
SEND;
```

In the above example, when the SAVE command is encountered, all the numbered statements that follow (6, 9, 18) will be stored in the PLAN DYNAMIC file 2 on drive 3. This is known as explicit saving because the statements are stored for execution at a later time, and not executed now. (See EXECUTE Command, discussed above). Implicit saving, is utilized where statement storage and execution are accomplished as the statements are read.

It is important to note that execution of the SAVED statements will occur by statement numeric sequence, not by position within the input SAVE stream. For example, if a statement number 15 was placed after statement 18 in the stream, it would still be executed ahead of 18 if at a later time an EXECUTE command was encountered utilizing the parameters FROM 9 and TO 18.

PHRASE TABLE DUMP

ALTER PHRASE: DUMP PHRASES, I(500) SYSTEM1130, I(501)NOD100, I(503)LEVEL1, LEVEL1, (200)"CHECKSUM", "PHRASE NAME", "LEVEL TYPE-OBJECT", "ENTRY SIZE", "VERB", "SUBSCRIPT NAME VALUE RANGE INDEX", "EXIT PROGRAM LIST", "SYMBOL EXIT FORMAT SCALE SUBSCRIPT EXPRESSION", "PROGRAM LIST", "TEST LOCATION ACTION", "LITERAL , LIST",SUBSCRIPT, "LOCATION MODE FACTOR EXPRESSION", (510)-*TP'CON DUM PHR I(504)DRIO';

ADD PHRASE: CON DUMP PHRASES, (281)"INTERPRETIVE EXPRESSIONS", "VERB PROGRAMS", END OF PHRASE TABLE DUMP", PROGRAM'DFJPTDMP', (505)"DFJPFILE", (835)NAM "DFJPTDP1DFJPTDP2DFJPTDP3DFJPTDP5 DFJPTDP6";

DUMP PHRASES is a command that produces a tabulation of the phrases that exist within PFILE.

CON DUMP PHRASES is a CONTINUTATION OF THE DUMP PHRASES command and should not be invoked by itself.

ALTER PHRASE: INPUT, I(-8)M, I(M)NOD1,0, LEVEL1, PROGRAM'DFJPIOCS';

ALTER PHRASE: OUTPUT,I(-8)M, I(M)0, I(M+1)NOD101, LEVEL1, PROGRAM 'DFJPIOCS';

The module DFJPTDMP produces the phrase dump. It requires XACES, XTRAC, XPRNT, and XBIT, which are called as subroutines. DFJPTDP1, DFJPTDP2, DFJPTDP3, DFJPTDP5, and DFJPTDP6 are also required. They are are loaded as PLAN system local modules on OS PLAN. These modules are special purpose programs that have no use in any other environment.

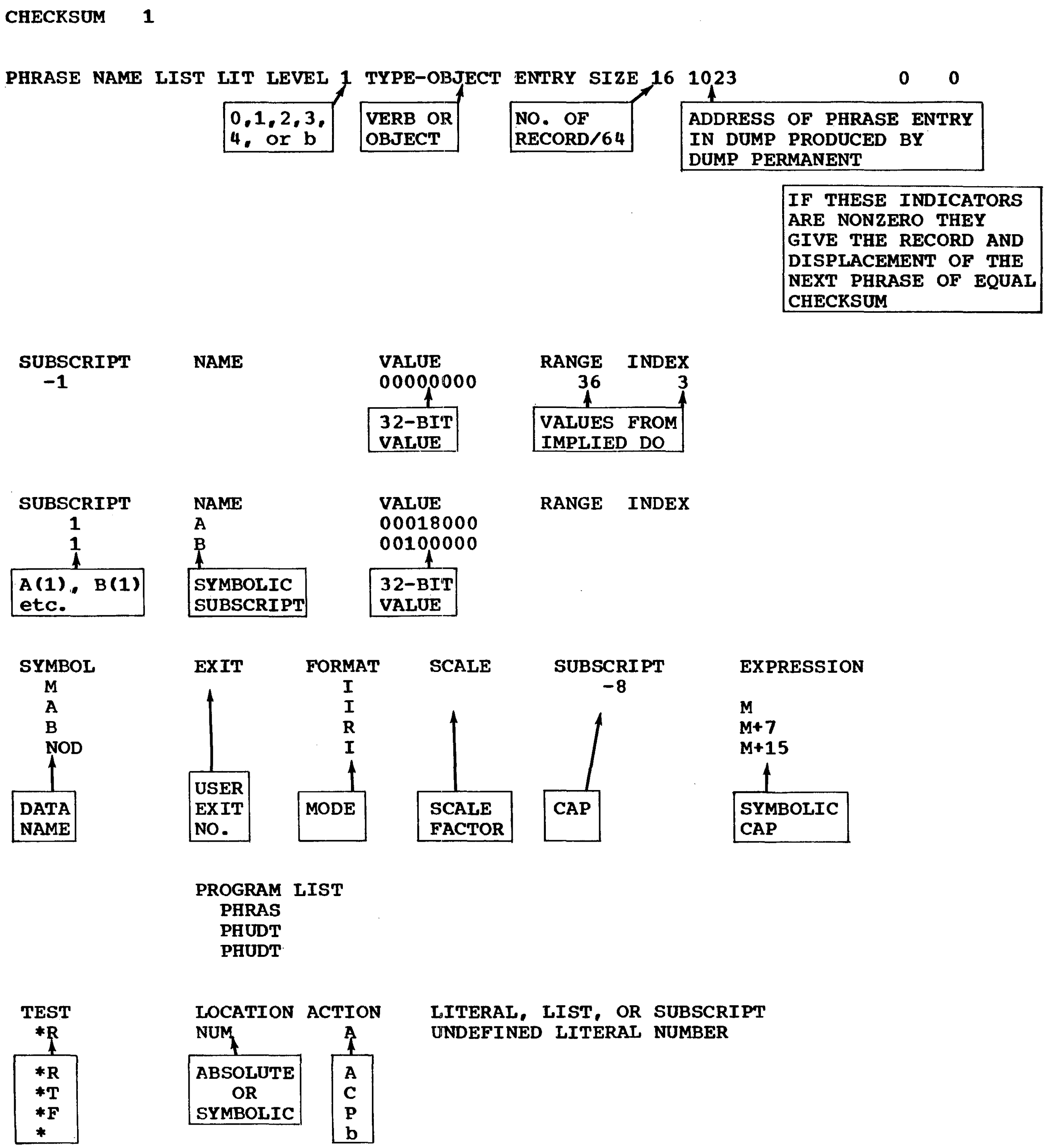
DUMP PHRASES FUNCTION	NAME	CAP	MODE	DEFAULT VALUES	CHECKING RULES	EXPRESSIONS
SYSTEM DESIGNATION	SYSTEM	500	I	360		
OUTPUT DEVICE	DEVICE	501	I	100		
PRINTOUT LEVEL	LEVEL	503	I	1		

1. SYSTEM DESIGNATION. This parameter defines the system for which the PFILE (PLAN language dictionary) is being dumped. The phrase for the appropriate system contains the necessary standard value so that the user should never be required to specify this parameter.

2. OUTPUT DEVICE. This parameter defines the sequential device code to be used for output.

3. PRINTOUT LEVEL. This parameter defines the complexity of the phrase listing to be produced. Each higher level incorporates all items of the lower levels.

The items listed below represent information that is produced at the various printout levels. Figure 8 shows sample lines from the dump. Enclosed items are explanatory notes about the sample output lines. It is strongly recommended that the reader make a deligent attempt to correlate the phrases as defined in this section with the listing produced with the DUMP PHR, LEVEL 6; command through use of Figure 8.



```
CHECKSUM    1

PHRASE NAME LIST LIT LEVEL 1 TYPE-OBJECT ENTRY SIZE 16 1023          0    0

 SUBSCRIPT      NAME         VALUE      RANGE  INDEX
   -1                        00000000     36       3

 SUBSCRIPT      NAME         VALUE      RANGE  INDEX
      1         A            00018000
      1         B            00100000

 SYMBOL         EXIT    FORMAT   SCALE    SUBSCRIPT      EXPRESSION
   M                      I                  -8
   A                      I                                M
   B                      R                                M+7
   NOD                    I                                M+15

                PROGRAM LIST
                  PHRAS
                  PHUDT
                  PHUDT

 TEST           LOCATION ACTION   LITERAL, LIST, OR SUBSCRIPT
  *R            NUM        A      UNDEFINED LITERAL NUMBER
```

0,1,2,3, 4, or b

VERB OR OBJECT

NO. OF RECORD/64

ADDRESS OF PHRASE ENTRY IN DUMP PRODUCED BY DUMP PERMANENT

IF THESE INDICATORS ARE NONZERO THEY GIVE THE RECORD AND DISPLACEMENT OF THE NEXT PHRASE OF EQUAL CHECKSUM

32-BIT VALUE

VALUES FROM IMPLIED DO

A(1), B(1) etc.

SYMBOLIC SUBSCRIPT

32-BIT VALUE

DATA NAME

USER EXIT NO.

MODE

SCALE FACTOR

CAP

SYMBOLIC CAP

*R
*T
*F
*

ABSOLUTE OR SYMBOLIC

A
C
P
b

Figure 8. Phrase table dump explanation

LEVEL	ITEM LISTED
0,1	Phrase name
	Phrase level
	Type (object or verb)
	Number of internal records (80-bit on 1130, 64-bit on System/360) required for phrase
	PFILE ADDRESS of phrase entry
	Chained phrase indicator (0 0 means no chained phrase)
	Checksum of phrase
2	Initialization (Default values)
	Subscript
	Name
	Value
	Range
	Index
3	Symbol Table
	Symbol
	User-exit number
	Format
	Scale factor
	Subscript
	Subscript expression
4	Program lists
5	Check entries
	Test
	Location
	Action
	Literal, list, or subscript
6	Expressions
	Data area
	Formula area

ERROR LISTING

ALTER PHRASE: DUMP ERRORS, PRO'DFJPEDMP';

DUMP ERRORS is a command that causes all diagnostics in the error queue file to be listed on the PLAN diagnostic device.

PAGE LENGTH DEFINITION

ALTER PHRASE: SET PAGE LENGTH, I(-8)M, I(M)PGL60, I(M+1)NOD100, PROGRAM'DFJPLENG';

SET PAGE LENGTH is a blank-level command that allows the user to specify the number of printed lines per page on a sequential device that is to contain printed output.

SET PAGE LENGTH FUNCTION	NAME	CAP	MODE	DEFAULT VALUES	CHECKING RULES	EXPRESSION
ERASABLE COMMON POINTER	M	(-8)	I			
PAGE LENGTH	PGL	(M)	I	60		
OUTPUT DEVICE	NOD	(M+1)	I	100		

1. PAGE LENGTH. This parameter defines the number of lines to be printed on a page before a logical EOF is generated and an automatic eject (skip to 1) is effected.

2. OUTPUT DEVICE. This parameter defines the sequential device code with which the PAGE LENGTH operand is to be associated.

ALTER PHRASE: INPUT, I(-8)M, I(M)NOD1,0, LEVEL 1, PROGRAM'DFJPIOCS';

INPUT is a command that may be issued to change the device that is assigned as the standard PLAN input device.

1. NOD. This parameter defines the number of the device that is to be used for PLAN input. The output device is not changed.

ALTER PHRASE: OUTPUT, I(-8)M, I(M)A0, I(M+1)NOD101, LEVEL1, PROGRAM'DFJPIOCS';

OUTPUT is a command that may be used to change the device that is assigned as the standard PLAN output device.

1. NOD. This parameter defines the number of the device that is to be used for PLAN output. The input device is not changed.

SPECIAL PURPOSE OS PHRASES

ALTER PHRASE: CREATE LOADER ENTRIES, PROGRAM'DFJLLIST';

CREATE LOADER ENTRIES is a command that gives OS PLAN the capability of referencing the RAM or LINKPAC areas.

The general format of this command is:

```
    CREATE LOADER ENTRIES:  (NAME1,....);
```

where NAME1,... is a load module name that is to be loaded into the partition via the LOAD macro and be made available as entry points for the execution of any loader call. This allows programs in the LINKPAC or RAM areas to be objects of a CALL LOCAL. The names specified in the LIST must be in the JOBLIB PDS.

The maximum number of names in the list is 75. Use of this command destroys any entries defined by previous use of the command.

Programs that reference blank COMMON may not be operands of this command.

ALTER PHRASE: CREATE CORE DIRECTORY, PROGRAM 'DFJCRDIR';

CREATE CORE DIRECTORY is a command that allows the user to build an in-core PDS directory of names of frequently loaded modules.

```
    CREATE CORE DIRECTORY:  (NAME1,...);
```

NAME1,... is a load module name that is placed in the in-core PDS directory to decrease load time for those modules. The names in the list must be entries in the PLANLIB PDS.

Use of this command will replace the previous directory. The maximum number of entries is 75 names.

PROGRAM DEBUGGING AND ABEND DUMPS

This section is intended to assist the programmer in diagnosing unusual conditions and/or program errors that may cause either the PLAN system to terminate a phrase execution or the operating system to ABEND.

PLAN TRACE FACILITY

If the PARM 'TRACE' is included in the EXEC statement, the PLAN system will print on the output device a trace listing of all modules that are entered for execution. The printed line has the following format:

```
NAME=xxxxxxxx PCB=xxxxxx EP=xxxxxx EL=xxx
RL=xxx
```

where:

NAME is the program name of the module about to be entered.

PCB is an address pointing to the program control block for the module being loaded.

EP is the module entry point address.

EL is a number specifying the execution level at which the module will be executed.

RL is the segment number in which the module was loaded.

The TRACE facility can be very helpful in showing logical program problems because it will produce a sequential listing of the programs that were executed. If an ABEND occurs, the name of the program last printed is generally the module that caused the termination.

ABEND DUMPS

Although the PLAN system attempts to prevent ABENDS, there are certain errors that PLAN cannot control. These errors will cause abnormal termination of the JOB step. In this event, the programmer will have to use the ABEND dump provided by OS in order to determine the trouble.

To interpret an ABEND dump of a PLAN job, the following information is of importance to the programmer:

1. Start of BLANK COMMON
2. Start of the managed COMMON array
3. Start of the nonmanaged COMMON array
4. Total current length of BLANK COMMON
5. PLAN switch words
6. Location, entry points, and lengths of programs currently resident in the PROGRAM AREA
7. Origin of the OS FREE STORAGE areas
8. Name of the program currently in execution

In interpreting an OS/360 ABEND dump, the user will find the following publication very helpful:

> IBM System/360 Operating System Programmer's Guide to Debugging (C28-6670)

LOCATING THE BLANK COMMON ARRAY

The BLANK COMMON array always starts at the beginning of the PLAN system partition. To locate this array, the user should find the address of the DFJPLAN problem program area. This will be found in the front portion of the ABEND dump. This is the starting core address of the PLAN system monitor and the address of the BLANK COMMON array. It is the reference point from which all other elements in the PLAN partition can be located.

LOCATING MODULES IN THE PROGRAM AREA

Figure 9 is a diagram of the use of the program area by PLAN to load a program.

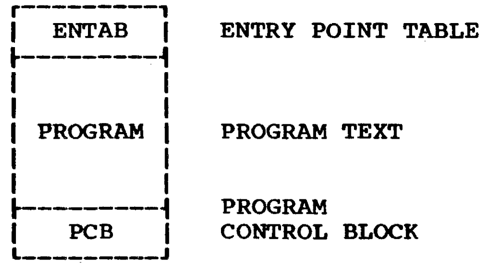


Figure 9. Use of the program area to load a program

Every program that is loaded by PLAN is preceded by an ENTAB which contains a listing of every CSECT name in the module and its actual core address. The format of an entry in this table is as follows:

BYTE	
0-7	CSECT NAME
8-11	CORE ADDRESS
12-15	LAST ENT IND

The PCB (Program Control Block) appended to the beginning of the program by the PLAN loader performs the same function as a PRB (Program Request Block) does for OS. It contains information that PLAN requires to maintain the program area. Its format is as follows:

0-7	MODULE NAME	
8-11	SEG. #	ENTAB ADDR
12-15	LENGTH OF COMMON	
16-19	MODULE ENTRY POINT	
20-24	NEXT PCB ADDR	

To locate any module in the program area, the user should find the start of the PCB chain, which is located at COMMON +14 hex and trace through the PCB chain. The last PCB has zeroes in the chain address. The address of the PCB for the current module in execution is located at COMMON +24.

When processing LOCAL's, the PLAN system saves the status of the calling program in a LOCAL control block. Its format is as follows:

BYTE	
0-63	GPR SAVE AREA STORED 14-13
64-67	CALLERS PARM ADDRESS
68-71	CALLER PICA ELEMENT
72-75	CALLERS PCB ADDRESS
76-83	MANAGED FREE STORAGE POINTERS
84-87	LOCAL CHAIN POINTER
88-93	CURRENT EXECUTION LEVEL
93-95	FILLER

To locate any LOCAL save area, find the start of the LCB chain at COMMON + 01C.

TABLE OF POINTERS IN PLAN COMMON

LOCATION	ITEM
COMMON + 150	Address of the PLAN system area
COMMON + 0A4	Address of the managed OS FREE STORAGE area
COMMON + 160	Address of the top of the PROGRAM AREA and the start of the nonmanaged OS FREE STORAGE area, if any
COMMON + 168	Address of the top of BLANK COMMON
COMMON + 198	Address of current entry in the program pop-up list
COMMON + 19C	Address of the end of the program pop-up list
COMMON + 024	Address of the PCB for the module currently in execution
COMMON + 000	Name of the last program loaded
COMMON + 014	Address of the last PCB in the PCB chain
COMMON + 9C4	PLAN Switch Words
COMMON + A00	Managed COMMON array
COMMON + 035	Current execution level
COMMON + 030	Address of the chain of LOCAL save areas
COMMON + 0F8	Address of the PLAN system save area
COMMON + 1B0	Columns 76-80 of the last command processed

PLAN SYSTEM DIAGNOSITC MESSAGES

This section contains a discussion of the control of diagnostic processing and lists diagnostic messages generated by various PLAN components through linkage to the error processor module "DFJPERRS". The format of PLAN system diagnostics is shown below:

```
DFJ000 001-100 TEXT
       101-200 TEXT
       201-300 TEXT
       301-400 TEXT
       401-450 TEXT
CCCnnn *A* mmmmm SEQ=yyy ID=ccccc
PG=xxxxxxxx DIAGNOSTIC
```

The segments of the diagnostic message underlined in the above example are variable. Functions defined by the variable data are:

TEXT This field of up to five lines contains the current input statement. It is printed only if the long-form diagnostic is requested. Character positions are printed to the left of the text.

CCC This three-character field is DFJ if the diagnostic is generated by PLAN and *** if generated by the user.

nnn This three-digit number is the error number assigned by the call to the error routines as calling parameter N1. In PLAN error diagnostics, this number is merely a diagnostic modifier (index).

A This character specifies the action taken following generation of the literal.

R indicates that execution of the current command is terminated. PLAN error recovery is initiated.

C indicates that the following generation of the diagnostic, the execution of the current command is terminated.

E indicates that the current execution of PLAN is terminated.

O indicates a pause for operator intervention.

mmmmm This is a five-digit modifier (ECODE) that provides additional information about the error. This parameter is provided as N2 in the call to the PLAN error subroutines.

SEQ=yyy This field provides the statement sequence of this PLAN statement relative to the beginning of the PLAN job stack.

ID=ccccc This five-character field provides the identification field (cc. 76-80) of the last card of the current PLAN statement.

PG=xxxxxxxx This field provides the name of the program in execution at the time the call to the error routine is issued.

DIAGNOSTIC This field contains the literal text of the diagnostic message and is limited to 76 characters.

PLAN ERROR PROCESSING

Since the PLAN system is a monitor which supervises the execution of other problem programs, it must have the ability to detect abnormal conditions.

There are four types of errors the PLAN system can detect and these are:

- Phrase Definition Errors
- Command Errors
- Execution Errors
- User-Defined Errors

1. Phrase definition errors are detected by the PLAN system module "DFJPHRAS" when a phrase is being entered into the PLAN language dictionary.

2. Command errors are detected by the PLAN system module "DFJPSCAN" while processing commands.

3. Execution errors are detected by the PLAN system mainline while a problem program is in execution.

4. User-defined errors are the result of a programmed call to one of the error

subroutines (ERROR, ERRET, ERREX, ERRAT).

Each type of error discussed is detected by a different module and at a different point in time. The technique used to produce a diagnostic in this environment may be described as follows: When an error is detected by any component of the system, the type of error is recorded and a generalized diagnostic processing module is called to produce the required error message. The PLAN system module that produces diagnostic messages is DFJPERRS.

The PLAN system offers the user several options in processing errors. Several terms are defined below that are used in describing these options.

SHORT FORM. The diagnostic message is produced without printing the phrase that caused it.

LONG FORM. The phrase that caused the diagnostic is printed with the diagnostic message.

IMMEDIATE MODE. The error processing module "DFJPERRS" is invoked at the time the error occurs, even if a checkpoint is required.

STACKED MODE. A condensed version of the error is recorded in the error message stack which will be processed the next time "DFJPERRS" is invoked by the system.

ERROR MSG STACK. An area on PFILE is reserved exclusively for recording errors in a condensed form. This gives the system the ability to delay calling the diagnostic processor "DFJPFRRS" until the program area is available.

ERROR MSG QUEUE. DYNAMIC file 255 on PLAN DYNAMIC drive 0 is reserved as a queue area for diagnostic messages. This gives the system the ability to post-list diagnostic messages by writing the messages on the file as they occur and then dumping the file on command.

USER-ERROR EXIT. The PLAN system has the ability to call a user-error processing module in the cases where the normal PLAN mode of diagnostic presentation is not appropriate for the application.

SPECIFYING ERROR PROCESSING MODE

The mode of error processing by the PLAN system is controlled by the PLAN Switch Words 11, 12, and 13. These switch words can be set by any PLAN command. The standard error processing mode is as follows:

1. Errors are stacked.
2. Error message format is short.
3. No error messages are queued for post-listing.
4. No user-error processing module will be called.
5. Messages are printed on the standard PLAN output device.
6. Errors detected by the PLAN DYNAMIC file routines cause a phrase abort.
7. Errors detected by the PLAN PERMANENT file routines cause a phrase abort.

Switch Words 11-13 are normally set by the following operands of the PLAN JOB command:

1. NERM
2. DEVICE
3. UMOD
4. SHORT
5. LONG
6. STACK
7. IMM
8. DFI
9. PFI

NERM specifies the number of error messages to be written on the error message queue file before they are dumped on the error message device.

DEVICE specifies the sequential file device code (NOD argument for PLINP/PLOUT subroutines) to which the diagnostic messages are to be written.

UMOD specifies the EBCDIC name of a user-error processing module to be called by the error processor "DFJPERRS" when an error is processed.

SHORT specifies that the SHORT form of the diagnostic is to be used when an error message is produced.

LONG specifies that the LONG form of the diagnostic message be used when an error message is produced.

STACK specifies that the system is to optimize error message processing by using the error message stack in PFILE to record messages until "DFJPERRS" can be called without a checkpoint.

IMM specifies that "DFJPERRS" is to be invoked at the time the error occurs.

DFI specifies that a phrase abort condition is not to occur on certain error conditions detected by the DYNAMIC file support subroutines.

PFI specifies that a phrase abort condition is not to occur on certain error conditions detected by the PERMANENT file support subroutines.

If both SHORT and LONG are specified, the LONG-form option is used. If both STACK and IMM are specified, the IMMEDIATE option is used.

Use of the operands PFI and DFI requires the application program to process the error conditions that would normally abort the PLAN statement. If these operands are specified and the required programming is not present, unpredictable results can occur. What generally takes place is the following: When the error is detected, the file control block is closed, and on the next reference to the file, an error message indicating an unopened file control block is issued. This masks the real reason for the error condition.

STANDARD ERROR PROCESSING

Normally, the PLAN system will process errors at SHORT form and in a stacked mode. The reason for using this technique is that the size of the PLAN error processing module is such that if the program area is not free, a checkpoint is required to load and execute DFJPERRS. Delaying the call to DFJPERRS until the program area is free eliminates the need for a checkpoint and so improves performance. The error message stack has a finite limit on the number of messages it can contain, and in cases where the stack overflows, a checkpoint is forced and DFJPERRS empties the stack.

POST-LISTING OF ERRORS

Some applications may require that error messages be suppressed until end of job. An example of this is a compiler, such as FORTRAN or COBOL, where the error messages are listed at the end of the compilation. The PLAN system provides this facility to the user as a standard option. In order to use this facility the PLAN system must have available PLAN DYNAMIC drive 0. DYNAMIC file 255 is used as an error message queue file. To invoke this facility the user must specify a value in system Switch Word 11.

The value in this switch word is used by the error processor "DFJPERRS" to determine the number of error messages to write on the error message queue file (drive 0, file 255) before dumping the file on an output device.

The message records on this file are written as 21-word or 124-character records. The first word of the record is an integer from -3 to +12, and is used as an argument for the PCCTL subroutine to effect carriage control for the data line that is contained in words 1-24 (characters 4-123). The data portion must be alphameric data in the A4 format. The data area of records produced by DFJPERRS contains the PLAN system diagnostic message text. The user may write records directly to this file from an application program by using the PLAN sub routine EWRIT.

The PLAN error message queue file is dumped on the diagnostic device under the following conditions:

1. The number of diagnostics messages added to the queue file exceeds NERM.

2. The subroutine ERLST is called.

3. The end of PLAN input (/*) is read by DFJPSCAN.

4. A level 0 phrase is processed.

5. A level 1 phrase is processed.

USER-ERROR EXIT PROCESSING

If a user module name is specified in system Switch Words 11 and 12, by specifying UMOD'NAME', the PLAN error processor DFJPERRS creates an arry DFJPERRS creates an array in 'erasable COMMON' that describes the error and then invokes the named module through the PLAN 'LOCAL' facility. This array is in the following format:

BYTE	CONTENTS
0-7	Program name issuing diagnostic call
8-11	Error number (N1 from error subroutine call)
12-15	Error code (N2 from error subroutine call)
16-20	ID from cc. 76-80
21	hexadecimal FF=system error, 0=user error
22	hexadecimal FF=abort, 0=continue
23	(unused)
24-27	Sequence
28-31	Length of literal in characters
32-107	Literal text
108-111	Character count of phrase
112-561	Phrase text

A program written as a user-error processor may not use the following PLAN subroutines ERROR, ERRAT, ERREX, ERRAT, ERLST, LREPT, LCHEX, LREPT and PUSH. Any error detected while a user-error processing module is in control causes cessation of all error processing.

The UMOD and the NERM or DEVICE specifications are mutually exclusive. Therefore, the automatic PLAN facility for post-listing of errors is not available, if a user-error processing module is used. The same effect may be produced, however, by using the subroutine EWRIT to create an error message queue file. A dump of the file may be forced by using the LIST subroutines to place the name "DFJPEDMP" into the pop-up list. This module will force a dump of the error message queue file and will also terminate the current statement.

PHRASE DIAGNOSTICS

The following group of diagnostics is generated from errors detected by DFJPHRAS (the ADD PHRASE processor). ECODE (m) for all diagnostics generated by DFJPHRAS is a pointer to the position at which the error condition was detected, except as otherwise noted. Position one is the first character of the command. The format of the descriptions of the diagnostics is:

- DIAGNOSTIC NUMBER(n), ACTION CODE, DIAGNOSTIC •
  REASON

- 21 *C* PHRASE TO DELETE CANNOT BE FOUND •
  A phrase that is to be deleted is not currently in PFILE. This can result from a DELETE PHRASE or an ALTER PHRASE. If it results from an ALTER PHRASE, the ADD PHRASE aspect of the command is not suppressed.

- 22 *R* NO ROOM TO ADD PHRASE •
  There is no contiguous vacant area in PFILE large enough to allow the current phrase to be added. PFILE must be reorganized, reestablished, or expanded.

  Usually, some space can be gained by reorganizing the file without changing its size. This is accomplished by deleting the phrases and then re-adding them.

  Additional space may be provided by enlarging PFILE if it is currently smaller than the maximum size. PFILE must be at least 14 records and not more than 268 records in length. This will also require that the phrases for the system be re-added

- 23 *R* PHRASE ALREADY DEFINED •
  An attempt to add a phrase that already exists in PFILE has been made. If the phrase to be added is a replacement for the existing phrase, the existing phrase must be deleted before the new phrase can be added.

- 24 *R* INVALID FORMAT IN PROGRAM LIST •
  A program list defined with the ADD (ALTER) PHRASE is found to contain invalid syntax. This can result from an unrecognizable numeric or special character.

- 25 *R* INVALID FORMAT IN USER-EXIT PROGRAM LIST •
  This error may result from:
  a. A program name not starting with an alphabetic character
  b. More than three programs in the list

  (Note that errors in the user-exit program list may also be diagnosed as error number 24.)

- 26 *R* KEYWORD ENTRY NOT TERMINATED BY COMMA OR SEMICOLON •
  A keyword (symbol table entry, PROGRAM, VERB, EXIT, or LEVEL) has been collected, but the keyword and associated data was not terminated with a comma or semicolon.

- 27 *R* LEVEL NUMBER GREATER THAN 4 •
  The number collected following the specification word LEVEL is greater than 4.

- 28 *R* NO SYMBOL DEFINED AFTER EXECUTION-DEFINED SYMBOL SUBSCRIPT EXPRESSION •
  A symbolic subscript expression requires a symbol (name) to be defined. The required symbol has not been found.

- 29 *R* CONSTANT SUBSCRIPT ZERO OR LESS THAN -15 •
  A constant subscript has been encountered that does not describe a valid location in the system switch words or communication array.

- 30 *R* IMPLIED DO SUBSCRIPT NOT FOLLOWED BY SINGLE-VALUED CONSTANT •
  The value following an implied DO subscript was not found to be a single-valued constant, that is, numeric, +, or -. This error can result from an implied DO subscript followed by:

  a. A literal default, that is, "ABC"
  b. No default value

- 31 *R* SYMBOL SUBSCRIPT GREATER THAN 8176 OR 511 WITH P-VALUE •
  A constant subscript that defines a symbol exceeds the maximum allowable value of 8176 without scale values (P values) or 511 after scale values.

- 32 *R* EXECUTION-DEFINED SYMBOL FOLLOWED BY IMPLIED SYMBOL •
  A symbol that is implied follows a symbol

associated with a symbolic (execution-defined) subscript. There may not be an implied symbol after a symbolic subscript.

• 33 *R* PHRASE DEFINITION INVALID •
A phrase is not defined properly, that is the phrase name is syntactically incorrect. This can be caused by:

a. Failure to end the phrase definition with a comma.
b. Use of nonalphabetic characters within the phrase definition.

• 34 *R* SUBSCRIPT FOR DATA VALUE GREATER THAN 16,368 •
A communication array subscript greater than 16,368 has been encountered.

• 35 *R* INVALID CHARACTER •
The ECODE pointer indicates a character that is invalid in a phrase definition. This error can result from an error within the phrase further to the left that was undetectable at that phase of the scan.

• 36 *R* BCD LEFT PARENTHESIS IN LOGICAL EXPRESSION •
All characters in a logical expression must be punched in the EBCDIC code.

• 37 *R* USER-EXIT NUMBER GREATER THAN 3 •
User exits must be 1, 2, or 3.

• 38 *R* FORMULA NUMBER USED BEFORE FORMULA BLOCK •
A conditional exit includes a formula number, but a $n introducing the expression area has not been encountered.

• 39 *R* FORMULA NUMBER ZERO OR GREATER THAN 1024 •
The valid range for formula numbers is from 0 to 1024 in a phrase definition.

• 40 *R* UNDEFINED FORMULA NUMBER IN FORMULA AREA •
A transfer type formula has been encountered that references a nonexistent formula number. Ecode is set to the formula number found to be in error.

• 41 *R* MULTIPLE DEFINITION OF FORMULA NUMBER IN FORMULA AREA •
More than one formula is identified with the same number within this phrase.

• 42 *R* INVALID FORMAT IN FORMULA AREA •
Formula numbers must be followed by:
a. Another formula number
b. Expression
c. Symbol
d. Semicolon
e. Comma

• 43 *R* P-VALUE GREATER THAN 7 •
A scale factor greater than plus seven or less than minus seven has been encountered.

• 44 *R* KEYWORD 'PROGRAMS' NOT FOLLOWED BY PROGRAM LIST •
A program specification has been processed, but a program list is missing. This can result from the next significant character not being a quotation mark.

• 45 *R* INVALID FORMAT IN RELATIONAL EXPRESSION •
A syntax error has been processed in a relational expression. Possible reasons for this error are:
a. Unbalanced parentheses
b. A semicolon invalid within (not at end of) an expression

• 46*R* PROGRAM NAME CONTAINS TOO MANY CHARACTERS •
The maximum allowable length for a program name is eight characters.

• 47 *R* SEMICOLON IN LITERAL OR EMPTY LITERAL •
A semicolon is an invalid literal character. This diagnostic may result from failure to include the terminal quotation mark of a literal. The phrase terminating semicolon may then appear to be within the literal. A zero-length literal is invalid.

• 48 *R* INVALID FORMAT IN SYMBOLIC SUBSCRIPT EXPRESSION •
The indicated position contains a character that forms an invalid context for a subscript (arithmetic) expression. These conditions include:
a. Adjacent arithmetic operators
b. Unmatched parenthesis
c. Invalid characters
d. Expression does not end with comma

• 49 *R* USER EXITS NOT ALLOWED ON NEGATIVE SUBSCRIPTS •
An attempt has been made to define a user exit to store data in the switch area.

• 50 *R* INVALID FORMAT IN LOGICAL OR ARITHMETIC EXPRESSION •
ECODE points to a character that may not be contained in the context of a logical or arithmetic expression. These conditions include:
a. Adjacent arithmetic operators
b. Unmatched parenthesis
c. Invalid characters
d. Expression does not end with comma

• 51 *R* INVALID FORMAT IN SUBSCRIPT EXPRESSION •
The indicated position contains a character that forms an invalid context for a subscript (arithmetic) expression.

These conditions include:
a. Adjacent arithmetic operators
b. Unmatched parenthesis
c. Invalid characters
d. Expression does not end with comma

- 52 *R* EXPRESSION SUBSCRIPT GREATER THAN 8176 OR 511 WITH P-VALUE •
  The symbolic subscript that is associated with a phrase-defined expression is greater than 8176 (if a scale factor is not defined) or greater than 511 (if a scale factor is defined).

- 58 *R* NUMBER OUTSIDE ALLOWABLE FLOATING-POINT RANGE •
  A number has been given that cannot be represented in the floating-point representation of the PLAN system.

- 64 *R* PHRASE ENTRY TOO LARGE •
  The total phrase size is greater than 1024 bytes and will not be added, or one of the eight internal phrase tables is longer than 512 bytes. ECODE is either the total size of the phrase or the PFILE internal table number that is too large.

- 65 *R* ILLEGAL SYMBOL - CANNOT BE 'E' •
  A data name has been defined to be E. E is not allowed because of syntactical confusion with the exponential indicator E.

- 66 *R* INVALID FORMAT IN IMPLIED DO SUBSCRIPT •
  A syntactical error has been encountered. Reasons for this diagnostic may be:
  a. The increment (I_3) is negative.
  b. The limit (I_2) is negative.
  c. The limit divided by the increment is not a whole number.
  d. (I_2) or (I_3) is not a numeric constant.

- 68 *R* LEG OF CONDITIONAL EXPRESSION NOT EXPRESSION OR FORMULA NUMBER •

  The TRUE action leg or FALSE action leg of a conditional expression is not an expression (example: ?=B*100) or a formula number (example: ?$5).

- 70 *R* CHECK-ENTRY SUBSCRIPT GREATER THAN 8176 •
  The constant subscript that is associated with a check entry is greater than 8176.

- 71 *R* INVALID FORMAT IN CHECK-ENTRY LITERAL •
  A check entry must be in the following format when the literal option is exercised:

  *A'LITERAL'
  *C'LITERAL'
  *RC(SUBSCRIPT)

  The following condition may have been detected:
  a. LITERAL in improper format
  b. Quotation marks unmatched
  c. A subscript greater than 16,383

- 72 *R* UNBALANCED PARENTHESIS IN PROGRAM LIST •
  An unequal number of left and right parentheses have been found in a program list.

- 80 *C* UNREFERENCED FORMULA NUMBER IN FORMULA AREA **UPDATE NOT SUPPRESSED** •
  The formula area has been found to contain a formula number that is not referenced in another expression. ECODE: Formula number that is unreferenced.

EXECUTION-TIME DIAGNOSTICS

The following errors are detected during execution of logic modules operating within the PLAN environment. All 100 series errors result in a PLAN "Phrase Abort" and subsequent level error recovery. The format of the definitions for this section is:

- NUMBER *ACTION CODE* DIAGNOSTIC •
  PROGRAM INDICATED
  ECODE MEANING
  REASON FOR ERROR

- 101 *R* PROGRAM NAMED NOT IN PLAN LIBRARY •
  Program: Program name not found
  ECODE: Unused.
  Reason: The named program was not found in the search of the PLAN library PDS.

- 102 *R* INVALID COMMON DEFINITION ENCOUNTERED •
  Program: Program name.
  ECODE: Unused.
  Reason: The length of COMMON for the named program is less than 640 FORTRAN (32-bit) words.

- 103 *R* PROGRAM TOO LARGE FOR AVAILABLE MEMORY •
  Program: Program name.
  ECODE: Unused.
  Reason: The size of the name program exceeds the size of the available area for program loading.

- 104 *R* PROGRAM NAME IN INVALID FORMAT •
  Program: '••••••••' (Unpredictable)
  ECODE: Unused.
  Reason: An invalid program name has been found in the pop-up list.

- 105 *R* PROGRAM FORMAT INVALID •
  Program: Program name.
  ECODE: Unused.

Reason: The named program is in overlay, scatter mode or contains TESTRAN symbol cards on OS PLAN.

- 110 *R* CHECKPOINT PROCESSING INVALID •
  Program: Last program entered.
  ECODE: Unused.
  Reason: a. An * was encountered in the pop-up list without a checkpoint being in effect.
  b. A checkpoint call when either there is no checkpoint file or insufficient room to write the complete checkpoint.

- 111 *R* OVER 50 NAMES IN POP-UP LIST •
  Program: Last program entered.
  ECODE: Unused.
  Reason: An attempt to place more than 50 names in the pop-up list has been made.

- 112 *R* LOCAL PROCESSING INVALID •
  Program: Program issuing CALL LOCAL.
  ECODE: Unused.
  Reason: a. There is not room to load The program called as a LOCAL.

- 113 *R* LSAV OR LRLD PROCESSING INVALID •
  Program: Program issuing loader call.
  ECODE: Unused.
  Reason: On System/360 all calls to LSAV or LRLD are invalid.

Note: In all 120-130 series diagnostics ID(1) is set to a closed status. Any further attempt to read or write to the file without reopening the file will result in a phrase abort, and PLAN level error recovery will be invoked.

- 120 *R* UNOPENED FILE CONTROL BLOCK ON CALL READ/WRITE •
  Program: Last program entered.
  ECODE: File number.
  Reason: ID(1) in the file control block is in a closed status.

- 122 *R* INVALID DRIVE CODE OR FILE CONTROL BLOCK ON CALL FIND/RELES •

  Program: Last entered.
  ECODE: Unpredictable.
  Reason: a. File number is zero.
  b. Drive code is negative or greater than 7.

- 123 *R* INVALID FILE CONTROL BLOCK ON CALL READ/WRITE •
  Program: Last program entered.
  ECODE: Unpredictable.
  Reason: a. ID(1) has been altered.
  b. The file specified by ID(1) has been released because of an allocation request for a higher-priority file.
  c. The file specified by ID(1) was automatically released because a phrase of higher priority than the file was processed. This can apply only to ID control blocks that reside in COMMON through phrase boundaries.

- 124 *R* INVALID KDIS/KOUNT ON CALL READ/WRITE •
  Program: Last program entered.
  ECODE: File number.
  Reason: KDIS or KOUNT is negative or KDIS+KOUNT exceeds maximum file size.

- 125 *R* DYNAMIC DRIVE NOT MOUNTED •
  Program: Last entered.
  ECODE: File number.
  Reason: A DYNAMIC drive required by a CALL FIND/READ/WRITE/RELES is not available to the system.

- 126 *R* INSUFFICIENT SPACE FOR ALLOCATION ON CALL FIND/WRITE •
  Program: Last entered.
  ECODE: File number.
  Reason: a. On a CALL FIND insufficient space is available to satisfy the NALLO argument.
  b. On a CALL WRITE insufficient space is available for secondary allocation.

- 130 *R* UNOPENED FILE CONTROL BLOCK ON CALL RDATA/WDATA •
  Program: Last program entered.
  ECODE: File number.
  Reason: ID(1) in the file control block was not initialized.

- 132 *R* INVALID DRIVE CODE OR FILE CONTROL BLOCK ON CALL GDATA •

  Program: Last entered.
  ECODE: Unpredictable.
  Reason: a. File number is zero.
  b. Drive code is negative or greater than 7.
  c. File name is not in PLAN library.

- 133 *R* INVALID FILE CONTROL BLOCK ON CALL RDATA/WDATA •
  Program: Last program entered.
  ECODE: Unpredictable.
  Reason: ID(1) has been altered

- 134 *R* INVALID KDIS/KOUNT ON CALL RDATA/WDATA •
  Program: Last program entered.
  ECODE: File number.
  Reason: KDIS or KOUNT is negative or

KDIS + KOUNT exceeds maximum file size.

- 135 *R* PERMANENT DRIVE NOT FOUND •
Program: Last program entered.
ECODE: File number.
Reason: The PERMANENT drive is not defined on a PLFSYnn DD card.

- 140 *R* INVALID RECORD LENGTH ON CALL PSORT/PMERG •
Program: DFJPSRTA/DFJPMERG
ECODE: File number.
Reason: Word 1 of the sort control list is minus or greater than 512.

- 141 *R* INVALID SORT CONTROL FIELD COUNT ON CALL PSORT/PMERG •
Program: DFJPSRTA/DFJPMERG
ECODE: File number.
Reason: The number of sort fields is specified as negative, zero, or greater than 99 or extends byeond the end of COMMON.

- 142 *R* INVALID SORT CONTROL FIELD ON CALL PSORT/PMERG •
Program: DFJPSRTA/DFJPMERG
ECODE: File number.
Reason:
  a. Word 1 of the sort control field is out of range (-6 to +6).
  b. Boundary alignment of displacement is invalid for type of sort.
  c. The sort field extends beyond the length of the record.
  d. The number of element specified is not a positive integer.

- 143 *R* INSUFFICIENT FILE SPACE TO EXECUTE PMERG FUNCTION •
Program: DFJPMERG
ECODE: Merge file number.
Reason: The required space for the output file of the merge is not available.

- 144 *R* INSUFFICIENT WORK AREA IN MANAGED AREA FILE FOR PSORT FUNCTION •
Program: DFJPSRTA
ECODE: File number.
Reason: Self-explanatory

- 145 *R* MERGE FILE OUT OF SEQUENCE ON CALL PMERG •
Program: DFJPMERG
ECODE: File number.
Reason: Self-explanatory.

- 146 *R* UNOPENED FILE CONTROL BLOCK ON CALL PSORT/PMERG •
Program: Program calling PSORT/PMERG
ECODE: File number.
Reason: The file control block specified is found not to be properly opened.

- 147 *R* FILE TO SORT DOES NOT EXIST •
Program: DFJPSRTA
ECODE: File number.
Reason: Specified file cannot be found on the drive specified in the file control block.

- 150 *R* INVALID RECORD LENGTH ON CALL GSORT/GMERG •
Program: DFJGSRTA/DFJGMERG
ECODE: Record length.
Reason: Word 3 of the sort control list is minus or greater than 512.

- 151 *R* INVALID SORT CONTROL FIELD COUNT ON CALL GSORT/GMERG •
Program: DFJGSRTA/DFJGMERG
ECODE: Sort field count.
Reason: The number of sort fields is specified as negative, zero, or greater than 98.

- 152 *R* INVALID SORT CONTROL FIELD ON CALL GSORT/GMERG •
Program: DFJGSRTA/DFJGMERG
ECODE: Sort control field sequence.
Reason:
  a. Word 1 of the sort control field is out of range (-6 to +6).
  b. Boundary aligment of displacement is invalid for type of sort.
  c. The sort field extends beyond the length of the record.
  d. The number of elements specified is not a positive integer.

- 153 *R* INSUFFICIENT FILE SPACE TO EXECUTE GMERG FUNCTION •
Program: DFJGMERG
ECODE: Merge file number.
Reason: The required space for the merged file is not available.

- 154 *R* INSUFFICIENT WORK AREA IN MANAGED AREA SAVE FILE FOR GSORT FUNCTION •
Program: DFJGSRTA
ECODE: File number.
Reason: Self-explanatory.

- 155 *R* MERGE FILE OUT OF SEQUENCE ON GMERG •
Program: DFJGMERG
ECODE: File number.
Reason: Self-explanatory.

- 156 *R* UNOPENED FILE CONTROL BLOCK ON CALL GSORT/GMERG •
Program: Program calling GSORG/GMERG.
ECODE: File number.

Reason: The file control block specified is found not to be properly opened.

- 171 *R* INVALID SAVED STATEMENT EXECUTION FILE •
  Program: DFJPSTSV
  ECODE: File number.
  Reason: The header of the indicated file is found not to be valid for a statement save file.

- 172 *R* STATEMENT TO EXECUTE NOT IN SAVE FILE •
  Program: DFJPSTSV
  ECODE: The number of the statement to be executed from the save file.
  Reason: A statement has been indicated for retrieval from the statement save file but cannot be found.

- 173 *R* PROGRAM ERROR IN SAVED STATEMENT RETRIEVAL •
  Program: DFJPSTSV
  ECODE: The invalid value causing the error.
  Reason: The saved statement file has been destroyed or overwritten.

- 180 *R* INVALID LITERAL FILE •
  Program: DFJPDIAG or DFJPLITL
  ECODE: The file number.
  Reason: A file defined for literal processing cannot properly be opened by GDATA.

PSCAN DIAGNOSTICS

The following diagnostics are generated as a result of errors detected by PSCAN while processing the phrases and language definition file (PFILE). Format of the diagnostic descriptions is the same as that for the ones in the preceding section.

- 201 *R* PHRASE SKIPPED •
  ECODE: Unused.
  Reason: DFJPSCAN has caused the statement to be bypassed because of an error in a preceding command upon which this command is dependent.
  Action: The next command is processed.

- 210 *R* LEVEL 0 PHRASE NOT ENCOUNTERED •
  ECODE: Cursor.
  Reason: A level 0 phrase was not encountered following the invoking of PLAN.
  Action: Statements are skipped until a level 0 phrase is encountered.

- 220 *R* LEVEL 1 PHRASE NOT ENCOUNTERED •
  ECODE: Cursor.
  Reason: The first recognizable command in a job stack depends logically on a statement that was not found. The preceding statement(s) may have resulted in a code 221 diagnostic.
  Action: Statements are skipped until a level 1 phrase is encountered. recog-nized

- 221 *R* UNDEFINED PHRASE •
  ECODE: Cursor.
  Reason: The command cannot be recognized in total or in part as a phrase defined in the systems dictionary. The statement scan is abandoned.
  Action: The scan of this command is terminated.

- 222 *R* STATEMENT OVER 450 CHARACTERS •
  ECODE: Cursor.
  Reason: A semicolon may be mispunched or missing.
  Action: Statement scan is terminated.

- 223 *R* PLAN WORD FALSE •
  ECODE: A subscript indicating the particular communication array location that was tested for not FALSE.
  Reason: The tested location was found to be FALSE.
  Action: Level error recovery and skipping is initiated

- 224 *R* PLAN WORD NOT REAL •
  ECODE: A subscript indicating the communication array location that was found to be TRUE or FALSE.
  Reason: A word required to be real is TRUE or FALSE.
  Action: Level error recovery and skipping is initiated.

- 225 *R* PLAN WORD NOT TRUE •
  ECODE: A subscript indicating the communication array location that was found to be FALSE or REAL.
  Reason: A word required to be TRUE is FALSE or REAL.
  Action: Level error recovery and skipping is initiated.

- 226 *R* PLAN WORD NOT FALSE •
  ECODE: A subscript representing the communication array that is found to be TRUE or REAL.
  Reason: A word required to be FALSE is found to be TRUE or REAL.
  Action: Level error recovery and skipping is initiated.

- 227 *R* UNDEFINED SYMBOL IN INPUT STREAM •
  ECODE: A cursor pointing to the end of the symbol in question.
  Reason: A symbolic data name has been misspelled, or a comma was

omitted after the command in a statement. No symbol table entry can be found for the word in this statement or in any statement upon which this statement is dependent. Failure to terminate a command with a semicolon results in the next command being interpreted as data for the command that precedes it.
Action: The command is not executed, but the scan is completed.

• 228 *R* UNDEFINED SYMBOL IN EXECUTION-DEFINED SYMBOL EXPRESSION •
ECODE: The sequence number of the expression in the phrase definition.
Reason: A symbolic subscript expression contains an undefined symbol.
Action: The scan is completed and the level error recovery is initiated.

• 229 *R* UNDEFINED SYMBOL IN PHRASE-DEFINED EXPRESSION •
ECODE: The sequence number of the expression in the phrase definition.
Reason: A symbol used in a phrase-defined expression is found to be undefined.
Action: The scan is completed and the level error recovery is initiated.

• 230 *R* OVER 8 VERBS IN INPUT STATEMENT •
ECODE: A pointer to the end of the ninth verb.
Reason: A command may not contain more than eight verb phrases and an object phrase.
Action: Statement scan is terminated.

• 231 *R* DITTO WORD IN COMMON NOT ALPHA •
ECODE: A pointer to the communication array word that is to be substituted in a command for a ditto mark.
Reason: Using the ditto character in a command depends on the definition of the preceding command. The word that is to be substituted is not alphabetic.
Action: The scan is terminated and level error recovery is initiated.

• 232 *R* EXECUTION-DEFINED SYMBOL SUBSCRIPT NOT POSITIVE •
ECODE: The sequence of the subscript expression within the phrase definition.
Reason: Evaluation of a symbolic subscript within the phrase definition has yielded a negative or zero result indicating an invalid communication array location.
Action: The scan is completed and level error recovery is initiated.

• 233 *R* EXECUTION-DEFINED SYMBOL SUBSCRIPT GREATER THAN 8176 OR 511 WITH P-VALUE •
ECODE: A number indicating the sequence of the symbolic subscript in the phrase definition.
Reason: The symbolic subscript expression, when evaluated, is found to be too large.
Action: The scan is completed and the level error recovery is initiated.

• 234 *R* INSUFFICIENT ROOM IN MANAGED ARRAY SAVE FILE •
ECODE: Number of additional words needed in PDATA file.
Reason: The file specified for saving the managed communication array is too small to allow saving of the context of the current managed array.
Action: The scan is completed and the level error recovery is initiated.

• 235 *R* MANAGED ARRAY DEFINITION TOO LARGE •
ECODE: The number of words in excess of the allowable size.
Reason: A communication array has been specified that cannot be accommodated by the current partition/machine size.
Action: The array is not saved or restored by PLAN data management, and the array is not initialized to FALSE at level 1 phrase time.

• 236 *R* INITIALIZATION VALUE SUBSCRIPT OUTSIDE OF COMMON •
ECODE: Value of subscript.
Reason: The CAP index for a default value is outside the current communication array.
Action: The value is not stored.

• 237 *R* DATA PLACEMENT FROM INPUT STREAM OUTSIDE OF COMMON •
ECODE: Input cursor.
Reason: The CAP index of an input value is outside the current communication array specification.
Action: The value is not written to the communication array.

• 239 *R* DATA PLACEMENT FROM PHRASE-DEFINED EXPRESSION OUTSIDE OF COMMON •
ECODE: Expression number.
Reason: The CAP index for storage of the results of an expression

evaluation is outside the current communication array specification.
Action: The value is not written to the communication array.

- 240 *R* FIRST CHARACTER IN INPUT STREAM AFTER PHRASE NOT COMMA, COLON, OR SEMICOLON •
ECODE: A cursor to the unexpected character.
Reason: The character required to start/terminate data collection was not encountered.
Action: The scan is completed and the level error recovery is initiated.

- 241 *R* UNRECOGNIZABLE CHARACTER IN INPUT STREAM •
ECODE: A cursor to the unrecognizable character.
Reason: A character cannot be interrogated in this context. It may have resulted from an illegal multipunch.
Action: The scan is completed and the level error recovery is initiated.

- 242 *R* SEMICOLON IN LITERAL OR EMPTY LITERAL •
ECODE: A cursor pointing to the invalid semicolon.
Reason: Either the literal closure character is missing or a semicolon is present within the literal.
Action: The scan is completed and level error recovery is initiated.

- 243 *R* NUMBER OUTSIDE ALLOWABLE FLOATING-POINT RANGE •
ECODE: A cursor to the end of the offending constant.
Reason: A number larger than can be contained in a floating-point number has been encountered.
Action: The scan is completed and level error recovery is initiated.

- 244 *R* IMPLIED DO NOT FOLLOWED BY SINGLE VALUED CONSTANT •
ECODE: A pointer to the position processed when the error was detected.
Reason: A single logical or numeric value does not follow an implied DO definition.
Action: The scan is completed and level error recovery is initiated.

- 245 *R* OVER 1000 EXPRESSION GO-TO'S EXECUTED •
ECODE: A number indicating the sequence of the expression found to be in error or input cursor.
Reason: Only 1000 formula GO-TO's are allowed within any phrase. This limit has been exceeded.
Action: The scan is completed and level error recovery is initiated.

- 246 *R* CHECK-ENTRY SUBSCRIPT OUTSIDE OF COMMON •
ECODE: Subscript value.
Action: The indicated communication array location is not checked.
Reason: The CAP index requiring execution of a check is outside the current communication array specification.

- 247 *R* DATA RETRIEVAL OUTSIDE OF COMMON
Program: PSCAN
ECODE: A cursor to the input stream subscript.
Reason: An attempt has been made to access data outside the current communication array.
Action: A 1.0 is supplied for arithmetic calculations and 0.0 for relational calculations. The scan is completed and level recovery is initiated.

- 248 *R* DATA RETRIEVAL OUTSIDE OF COMMON IN EXECUTION-DEFINED SYMBOL EXPRESSION •
ECODE: The expression number.
Reason: An attempt has been made to access data outside the current communication array.
Action: A 1.0 is supplied for arithmetic calculations and 0.0 for relational calculations. The scan is completed and level recovery is initiated.

- 249 *R* DATA RETRIEVAL OUTSIDE OF COMMON IN PHRASE-DEFINED EXPRESSION •
ECODE: The expression number.
Reason: An attempt has been made to access data from a location outside the current communication array specification.
Action: A 1.0 is supplied for arithmetic calculations and 0.0 for relational calculations. The scan is completed and level recovery is initiated.

- 255 *R* STATEMENT SAVE INVALID, PHRASE PUSHED FROM CHECK-ENTRY •
ECODE: CAP location being checked.
Reason: Implicit statement saving may not be combined with check entry pushed phrases.
Action: The statement is not saved; the PLAN error recovery is initiated, but the phrase is pushed.

- 263 *R* INVALID FORMAT IN INPUT STREAM EXPRESSION •
ECODE: A cursor to the offending

position.
Reason: An input stream expression is found to contain improper syntax. Reasons for this diagnostic may be:
a. Adjacent arithmetic operators
b. Operators following parenthesis
c. Parenthesis following operators
d. Invalid characters
Action: The scan is completed and level error recovery is initiated.

• 264 *R* INVALID FORMAT IN EXECUTION-DEFINED SYMBOL EXPRESSION •
ECODE: A number indicating the sequence of the expression in error.
Reason: A syntax error has been detected in the symbolic subscript defined at ADD PHRASE time. Reasons for this diagnostic may be:
a. Adjacent arithmetic operators
b. Operators following parenthesis
c. Parenthesis following operators
d. Invalid characters
Action: The scan is completed and level error recovery is initiated.

• 265 *R* INVALID FORMAT IN PHRASE-DEFINED EXPRESSION •
ECODE: A number indicating the sequence of the expression in error.
Reason: A syntax error has been detected in the phrase definition of an expression. Reasons for this diagnostic may be:
a. Adjacent arithmetic operators
b. Operators following parenthesis
c. Parenthesis following operators
d. Invalid characters
Action: The scan is completed and level error recovery is initiated.

• 266 *R* BCD LEFT PARENTHESIS USED IN INPUT STREAM LOGICAL EXPRESSION •
ECODE: A pointer to the erroneous parenthesis.
Reason: All logical expressions must be punched in EBCDIC code.
Action: The scan is completed and level error recovery is initiated.

• 268 *R* BCD LEFT PARENTHESIS USED IN PHRASE-DEFINED LOGICAL EXPRESSION •
ECODE: A number indicating the sequence number of the expression in error.
Reason: Logical expressions must be punched in EBCDIC code.
Action: The scan is completed and level error recovery is initiated.

• 269 *R INPUT STREAM EXPRESSION TOO COMPLICATED TO BE ANALYZED •
ECODE: A pointer to the position at which error was detected.
Reason: Too many levels of parenthesis have been encountered.
Action: The scan is completed and level error recovery is initiated.

• 270 *R* EXECUTION-DEFINED SYMBOL EXPRESSION TOO COMPLICATED TO BE ANALYZED •
ECODE: A number indicating the sequence of the expression found to be in error.
Reason: Too many levels of parenthesis have been encountered.
Action: The scan is completed and level error recovery is initiated.

• 271 *R* PHRASE-DEFINED EXPRESSION TOO COMPLICATED TO BE ANALYZED •
ECODE: A number indicating the sequence of the expression found to be in error.
Reason: Too many levels of parenthesis have been encountered.
Action: The scan is completed and level error recovery is initiated.

• 272 *R* INVALID FORMAT IN INPUT STREAM LITERAL RELATIONAL EXPRESSION •
ECODE: A pointer to the character processed when the error was discovered.
Reason: A syntax error in an alphabetic relational expression. This diagnostic may result from expressions of the nature:
a. 5="A"
b. A>"B"
c. B<"C"
Action: The scan is completed and level error recovery is initiated.

• 274 *R* INVALID FORMAT IN PHRASE-DEFINED LITERAL RELATIONAL EXPRESSION •
ECODE: A number indicating the sequence of the expression causing the error.
Reason: A syntax error in a phrase-defined relational. This diagnostic may result from expressions of the nature:
a. 5="A"
b. A>"B"
c. B<"C"
Action: The scan is completed and level error recovery is initiated.

• 275 *R* INVALID FORMAT IN INPUT STREAM SUBSCRIPT EXPRESSION •
ECODE: A pointer to the character processed when error was

detected.

Reason: A syntax error in a symbolic subscript or a subscript expression evaluation yields a negative result. Reasons for this diagnostic may be:
a. Result of subscript expression is not positive.
b. A logical value was encountered during the evaluation
c. An Implied Do was encountered in the evaluation of a subscript expression.

Action: The scan is completed and level error recovery is initiated.

• 276 *R* INVALID FORMAT IN EXECUTION-DEFINED SYMBOL SUBSCRIPT EXPRESSION •

ECODE: A pointer to the character processed when the error was detected.

Reason: A syntax error in symbol expression.

Action: The scan is completed and level error recovery is initiated.

• 277 *R* INVALID FORMAT IN PHRASE-DEFINED SUBSCRIPT EXPRESSION •

ECODE: A number indicating the sequence of the expression found to be in error.

Reason: A syntax error.

Action: The scan is completed and level error recovery is initiated.

• 278 *R* UNBALANCED PARENTHESES IN INPUT STREAM EXPRESSION •

ECODE: A pointer to the position at which the error was detected.

Reason: An unequal number of right and left parentheses are found in an expression.

Action: The scan is completed and level error recovery is initiated.

• 279 *R* UNBALANCED PARENTHESES IN EXECUTION-DEFINED SYMBOL EXPRESSION •

ECODE: A pointer to the position at which the error was detected.

Reason: An unequal number of left and right parentheses are found in an expression.

Action: The scan is completed and level error recovery is initiated.

• 280 *R* UNBALANCED PARENTHESES IN PHRASE-DEFINED EXPRESSION •

ECODE: A number indicating the sequence of the expression found to be in error.

Reason: An unequal number of left and right parentheses have been found, or a right parenthesis has been found with no preceding matched left parenthesis.

Action: The scan is completed and level error recovery is initiated.

• 281 *R* INVALID FORMAT IN INPUT STREAM CONDITIONAL EXPRESSION •

ECODE: A pointer to the position at which the error was detected.

Reason: A syntax error. Reasons for this diagnostic may be:
a ? or ! not followed by #, =, :, or $

Action: The scan is completed and level error recovery is initiated.

• 283 *R* INVALID FORMAT IN PHRASE-DEFINED CONDITIONAL EXPRESSION •

ECODE: A number indicating the sequence of the expression found to be in error.

Reason: A syntax error. Reasons for this diagnostic may be:
a ? or ! not followed by #, =, :, or $

Action: The scan is completed and level error recovery is initiated.

• 284 *R* INVALID FORMAT IN INPUT STREAM RELATIONAL EXPRESSION •

ECODE: A pointer to the position at which the error was detected.

Reason: A snytax error. Reasons for this diagnostic may be:
a. Unbalanced parenthesis
b. Invalid characters

Action: The scan is completed and level error revovery is initiated.

• 286 *R* INVALID FORMAT IN PHRASE-DEFINED RELATIONAL EXPRESSION •

ECODE: A number giving the sequence of the expression found to be in error.

Reason: A syntax error. Reasons for this diagnostic may be:
a. Unbalanced parenthesis
b. Invalid characters

Action: The scan is completed and level error recovery is initiated.

• 287 *R* INVALID END TO AN INPUT STREAM EXPRESSION •

ECODE: Input cursor.

Reason: An expression must end with a semicolon or comma.

Action: The scan is completed and level error recovery is initiated.

• 289 *R* INVALID END TO A PHRASE-DEFINED EXPRESSION •

ECODE: Sequence number of the expression in error.

Reason: An expression must end with a semicolon or comma.

Action: The scan is completed and level error recovery is initiated.

• 290 *R* LOGICAL EOF ENCOUNTERED IN PSCAN INPUT •

ECODE: Undefined.

Reason: A logical EOF has been set by a PSCAN CALL PLINP operation.

Action: The scan is completed and level error recovery is initiated.

- 299 *R or C* ******************** •
  ECODE: A pointer to the communication array upon which an unsuccessful test was made.
  Reason: The text for this diagnostic is normally user-defined text from a phrase-defined check entry. If the asterisks are provided, an error has been detected in the defined literal.
  Action: The phrase is terminated.

OS ONLY DIAGNOSTICS

The following messages are generated from the DD card edit performed by OS/360 PLAN. The message form is DDNAME, TEXT.

- 901 *E* XXXXXXXX NOT FOUND IN THE PLANLIB PDS •
  Program: PLAN
  Reason: The named module could not be loaded by the PLAN system. The modules are DFJPERRS, DFJPSCAN, or DFJRETN
  Action: PLAN execution is inhibited.

- 902 *E* DDNAME, DOES NOT SPECIFY A DIRECT ACCESS DEVICE •
  Program: PLAN
  Reason: The unit parameter of the specified DD card is incorrect.
  Action: PLAN execution is inhibited.

- 903 *E* DDNAME, DATA SET DOES NOT EXIST •
  Program: PLAN
  Reason: The data set named in the DD card does not exist on the specified volume.
  Action: PLAN execution is inhibited.

- 904 *E* DDNAME, INVALID BLKSIZE SPECIFICATION •
  Program: PLAN
  Reason: The specified BLKSIZE parameter is either too large for the unit specified or not a multiple of LRECL.
  Action: PLAN execution is inhibited.

- 905 *E* DDNAME, INVALID DSCB SPECIFICATIONS •
  Program: PLAN
  Reason: The data set named in the specified DD card:
  a. Has a partitioned data set format
  b. Has RECFM other than F or FB
  c. contains keys
  d. was never closed
  Action: PLAN execution is inhibited.

- 906 *E* DDNAME, INVALID SPACE ALLOCATION •
  Program: PLAN
  Reason: The space parameter in the named DD card does not specify TRK or CYL allocations.
  Action: PLAN execution is inhibited.

- 907 *E* DDNAME, I/O ERROR WHILE FORMATTING •
  Program: PLAN
  Reason: Input/Output error.
  Action: PLAN execution is inhibited.

- 908 *E* DDNAME, IS AN INVALID PLAN DD CARD •
  Program: PLAN
  Reason: The numeric specification on a PLINPxxx, PLOUTxxx, PLANDRVx, or PLFSxxxx DD card is invalid.
  Action: PLAN execution is inhibited.

- 909 *E* DDNAME, DATA SET INITIALIZED INCORRECTLY •
  Program: PLAN
  Reason: A PLANDRVx, PLSYSTAB, or PLNUMTAB was specified with DISP=OLD, and is not formatted correctly.
  Action: PLAN execution is inhibited.

- 910 *E* DDNAME, INSUFFICIENT FILE SIZE •
  Program: PLAN
  Reason: PLSYSTAB or PLANDRVx is not allocated sufficient space for correct execution.
  Action: PLAN excution is inhibited.

- 911 *E* DDNAME, NOT DEFINED IN A DD CARD •
  Program: PLAN
  Reason: PLSYSTAB or PLANLIB DD cards are missing.
  Action: PLAN execution is inhibited.

The following messages are generated from OS/360 PLAN during the initialization phase.

- 922 *E* XXXXXXX PARAMETER OR OPERAND IS INVALID •
  Program: PLAN
  Reason: The named parameter in the EXEC control card is invalid.
  Action: PLAN execution is inhibited.

  *NOTE: This message is printed on the system console device.

- 940 *R* DDNAME I/O ERROR •
  Program: Current program in control.
  Action: Phrase abort.

- 941 *R* XXXXXXXXXXXXXXXX •
  Program: Current program in control.
  Reason: A program interrupt has

occurred in a problem program. The diagnostic message is the program interrupt PSW.
Action: PLAN level error recovery is initiated.

- 942 *R* INSUFFICIENT PROGRAM AREA FOR PLAN FUNCTION •
  Program: PLAN
  Reason: The area allocated for the program area is too small.
  Action: Phrase abort.

- 999 *E* PLAN EXECUTION INHIBITED •
  Program: PLAN
  Reason: This action results if any of the error conditions listed occur.
  Action: PLAN execution is inhibited.

PLAN will ABEND during PLAN initialization with the following user codes:

- ABEND USER CODE 100 •
  Program: PLAN
  Reason: Missing or invalid PLINP/PLOUT DD card.
  Action: PLAN execution is inhibited.

- ABEND USER CODE 101 •
  Program: PLAN
  Reason: Unable to load one of the following PLAN modules: DFJLODER, DFJTRACE
  Action: PLAN execution is inhibited.

- ABEND USER CODE 102 •
  Program: PLAN
  Reason: No DD card supplied.
  Action: PLAN execution is inhibited.

- ABEND USER CODE 103 •
  Program: PLAN
  Reason: Insufficient core storage to initialize PLAN.
  Action: PLAN execution is inhibited.

GENERATING A PLAN SYSTEM

To generate a PLAN system the user must invoke the proper OS utility to load the four data sets from the supplied tape. The 'IEHMOVE' utility must be used to move PLAN.MODLIB and PLAN.SUBLIB (File sequence 3 and 4) to a direct access device. The 'IEBPTPCH' utility may be used to print or punch the sample problem (File sequence 1) and the standard phrases (File sequence 2).

The example below of Step 1 shows the use of IEHMOVE to load the two data sets (PLAN.MODLIB and PLAN.SUBLIB) from a magnetic tape (800 bpi) onto a 2311 disk volume. The 2311 is assumed to have enough space available to support the two data sets.

The SYSUT1 DD statement defines the device that is to contain the required IEHMOVE work data sets.

The DDB DD statement defines the receiving volume.

The TAPE DD statement defines the source volume.

The SYSIN DD statement defines the IEHMOVE control card input data set.

The MOVE statements cause the IEHMOVE program to move the unloaded data sets onto the receiving volume. See publication OS/360 Utilities (C28-6586) for further descriptions and examples of the use of the IEHMOVE utility.

The user should prepare JCL similar to the example shown but tailored to his individual installation requirements and then run the job step to move the data sets.

STEP 1 Retrieving the Libraries

```
//MOVE      JOB   84803,'JOE E. JONES',MSGLEVEL=1
//STEP      EXEC  PGM=IEHMOVE
//SYSPRINT  DD    SYSOUT=A
//SYSUT1    DD    UNIT=2311,VOLUME=SER=XXXXXX,DISP=OLD
//DDB       DD    UNIT=2311,VOLUME=SER=NEWPAC,DISP=OLD
//TAPE      DD    UNIT=2400,VOLUME=SER=PDT,LABEL=(3,NL),DISP=(OLD,PASS),   X
//                DSNAME=TAPE,DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSIN     DD    *
 COPY PDS=PLAN.MODLIB,TO=2311=NEWPAC,FROM=2400=(PDT,3),FROMDD=TAPE
 COPY PDS=PLAN.SUBLIB,TO=2311=NEWPAC,FROM=2400=(PDT,4),FROMDD=TAPE
/*
```

After successful execution of the move step the data set name PLAN.MODLIB should be entered into the system catalog data set (SYSCTLG). Below is a sample job stream to do this:

```
//CATLG        JOB  84803,'JOE E. JONES',MSGLEVEL=1
//             EXEC PGM=IEHPROGM
//DDA          DD   UNIT=2311,VOLUME=SER=111111,DISP=OLD
//SYSPRINT     DD   SYSOUT=A
//SYSIN        DD   *
   CATLG   DSNAME=PLAN.MODLIB,VOL=2311=NEWPAC
/*
```

For those systems using MVT the PLAN module IGG019WY is an EXCP appendage routine that must be in SYS1.SVCLIB for successful PLAN execution. The job stream listed below is required to place this module into the SVC library. It is assumed that PLAN.MODLIB is catalogued.

```
//LINK     JOB   84803,'JOE E.JONES',MSGLEVEL=1
//STEP     EXEC  PGM=IEWL,PARM='NCAL,LIST,RENT,LET'
//SYSUT1   DD    UNIT=SYSDA,SPACE=(CYL,(1,1))
//SYSPRINT DD    SYSOUT=A
//SYSLIB   DD    DSNAME=PLAN.MODLIB,DISP=OLD
//SYSMOD   DD    DSNAME=SYS1.SVCLIB,DISP=OLD
//SYSLIN   DD    *
  INCLUDE        SYSLIB(IGG019WY)
  ENTRY          IGG019WY
  NAME           IGG019WY(R)
/*
```

To add the standard PLAN commands to a PLAN phrase dictionary the user must execute a PLAN job. The standard PLAN commands are contained in file sequence 2 on the distribution tape. Below is a job stream to add the commands to a phrase dictionary:

```
//ADDPHR JOB   84803,'JOE E. JONES' MSGLEVEL=1
//JOBLIB DD    DSNAME=PLAN.MODLIB,DISP=OLD
//STEP   EXEC  PGM=DFJPLAN
//PLSYSTAB DD  DSNAME=DFJPFIL,UNIT=2311,VOLUME=SER=NEWPAC,              X
//             DISP=(NEW,CATLG),SPACE=(CYL,(5))
//PLOUT100 DD  SYSOUT=A
//PLINP001 DD  UNIT=2400,VOLUME=SER=PDT,DISP=(OLD,PASS),LABEL=(2,NL),   X
//             DSNAME=INPUT.PHRASES,DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//PLANLIB  DD  DSNAME=PLAN.MODLIB,DISP=OLD
/*
```

The JOBLIB DD statement specifies the library containing the PLAN system modules.

The PLSYSTAB DD statement defines a data set that will be a PLAN phrase dictionary. The name DFJPFIL is used but any other name is suitable. The DISP parameter specifies CATLG. This is required for successful execution of the sample problem.

The PLOUT100 DD statement specifies a data set for printed output from the PLAN system.

The PLINP001 DD statement defines the input data set for the PLAN system. In this case it is file number 2 on the distribution tape.

The PLANLIB DD statement defines the library PDS the PLAN system will use to load and execute modules.

If the user wishes to get the standard PLAN commands into cards the following job step should be run:

```
//PUNCH    JOB    84803,'JOE E. JONES',MSGLEVEL=1
//STEP     EXEC   PBM=IEBPTPCH
//SYSPRINT DD     SYSOUT=A
//SYSUT1     DD   UNIT=2400,VOLUME=SER=PDT,DISP=(OLD,PASS),LABEL=(2,NL),    X
//              DSNAME=INPUT.PHRASES,DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//SYSUT2     DD   UNIT=SYSCP
//SYSIN      DD   *
  PUNCH      TYPORG=PS
/*
```

The user may now run the sample problem. Refer to Appendix A in this manual.

OPTIONAL MACHINE-READABLE MATERIAL

This section provides an index to the optional program package. The tape supplied is an unlabeled tape containing two data sets as listed below:

FILE SEQUENCE 1

This data set, named PLAN.SRCLIB, is an unloaded version of a PDS. It was created by the OS utility IEHMOVE and contains the PLAN system source decks for both the PLAN system routines and the PLAN subroutine library. The direct access space required for this data set is approximately 650-1316 tracks. See Appendices C and D for a listing and description of the modules in this data set. The characteristics of this data set when moved to a direct access device will be RECORD LENGTH=80, BLOCK SIZE=400.

FILE SEQUENCE 2

This data set, named PLAN.MACLIB, is an unloaded version of a PDS. It was created by the OS utility IEHMOVE and contains the macro definitions necessary to assemble any PLAN system component.

The direct access space required for this data set is approximately 15-1316 tracks. See Appendix E for a listing and description of the members in this data set. The characteristics of this data set when moved to a direct access device will be RECORD LENGTH=80, BLOCK SIZE=3360.

The user must move the data sets PLAN. SRCLIB and PLAN.MACLIB to direct access devices using the OS utility IEHMOVE. The following is an example of a job stream to load these data sets to direct access devices.

```
//MOVE      JOB   84803,'JOE E. JONES',MSGLEVEL=1
//STEP      EXEC  PGM=IEHMOVE
//SYSPRINT DD     SYSOUT=A
//SYSUT1    DD    UNIT=2311,VOLUME=SER=XXXXXX,DISP=OLD
//DDB       DD    UNIT=2311,VOLUME=SER=NEWPAC,DISP=OLD
//TAPE      DD    UNIT=2400,VOLUME=SER=PDT,LABEL=(,NL),DISP=(OLD,PASS), X
//                DSNAME=TAPE,DCB=(RECFM=FB,LRECL=80,BLKSIZE=800)
//SYSIN     DD    *
 COPY PDS=PLAN.SRCLIB,TO=2311=NEWPAC,FROM=2400=(PDT,1),FROMDD=TAPE
 COPY PDS=PLAN.MACLIB,TO=2311=NEWPAC,FROM=2400=(PDT,2),FROMDD=TAPE
/*
```

The user may then print or punch any member of these data sets or he may assemble any module in the system using these data sets directly. Below is a sample job stream using these data sets directly. It is assumed that PLAN.SRCLIB and PLAN.MACLIB are catalogued data sets.

```
//ASMB        JOB  84800,'JOE E. JONES',MSGLEVEL=1
//STEP        EXEC PGM=IEUASM
//SYSPRINT    DD   SYSOUT=A
//SYSUT1      DD   UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSUT2      DD   UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSUT3      DD   UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSLIB      DD   DSNAME=SYS1.MACLIB,DISP=OLD
//            DD   DSNAME=PLAN.MACLIB,DISP=OLD
//SYSPUNCH    DD     UNIT=SYSCP
//SYSIN       DD   DSNAME=PLAN.SRCLIB(PLOUT),DISP=OLD
/*
```

NOTE: The assembler requires that concatenated data sets for SYSLIB have the same attributes.

APPENDIX A: RUNNING THE SAMPLE PROBLEM

The first file on the distribution tape contains the PLAN commands to execute the sample problem.

Before running the sample problem, insure that the following items have been done:

1. Check the output from the PLAN system MOVE step. There should be no error messages.
2. The standard PLAN phrases must be added to the dictionary file.

To run the sample problem the user should prepare the following JCL and run it as an OS job step. It is assumed that the data sets PLAN.MODLIB and DFJPFIL are cataloged.

```
//SAMPLE JOB    84803,'JOE E. JONES',MSGLEVEL=1
//JOBLIB DD     DSNAME=PLAN.MODLIB,DISP=OLD
//STEP   EXEC   PGM=DFJPLAN
//PLSYSTAB DD   DSNAME=DFJPFIL,DISP=OLD
//PLOUT100 DD   SYSOUT=A
//PLINP001 DD   UNIT=2400,VOLUME=SER=PDT,DISP=(OLD,PASS),LABEL=(,NL),    X
//              DSNAME=INPUT.SAMPLE,DCB=(RECFM=F,LRECL=80,BLKSIZE=80)
//PLANLIB  DD   DSNAME=PLAN.MODLIB,DISP=OLD
/*
```

EXPLANATION OF SAMPLE PROBLEM

The output of the sample problem as shown in this manual reflects the results of a PLAN system generation that includes initialization of a language definition dictionary file named DFJPFIL and the addition of the supplied standard phrases to the dictionary.

Card SMP01 is a PLAN JOB command which sets the length of the managed array to 510 words, specifies that an array beginning at word 200 of the managed array may be used for ERASABLE COMMON, and specifies that long-form diagnostics are to be produced in the event of an error.

This card accomplishes PLAN job initialization functions and satisfies the requirement that a level 0 command be the first command processed.

Card SMP02 contains the PLAN command LON and DUMP. The LON command is simply a dummy level 1 command that satisfies the requirement that a level 1 command immediately follow a level 0 command.

Note in the output listing (Figure 2), cards SMP01 and SMP02 are listed 80-80. The OS PLAN system normally lists all input commands. The DUMP command on card SMP02, which dumps the Switch Words, managed array and nonmanaged array, produces the output shown in Figure 3.

Noting the Switch Words the user can see that they contain the following:

SWITCH WORD	DECIMAL	HEX
8	200	C8
9	1150	47E
10	510	1FE
12	100	64
13	1	1

These values were placed in the Switch Words by the PLAN JOB command.

Card SMP03 is a DUMP PHRASES command which prints out the contents of the PLAN phrase dictionary. This card is listed in Figure 3. The output from this command is shown in Figure 4.

Card SMP04 is an ADD PHRASE command that will add the phrase SAMPLE TEST. This phrase shows an example of using a check entry test to produce a diagnostic message.

Card SMP05 is the SAMPLE TEST command. Cards SMP04 and SMP05 are shown listed in Figure 5.

Figure 6 shows the result of the SAMPLE TEST command. A long-form diagnostic is produced by the PLAN module DFJPSCAN and is the literal that was specified in the phrase SAMPLE TEST.

Card SMP06 is a DELETE PHRASE command to remove the phrase SAMPLE TEST from the dictionary.

When the PLAN command interpreter DFJPSCAN reads the end-of-file it returns control to the OS supervisor.

```
PLAN JOB,ERASABLE200,MANAGED510,LONG;                                    SMP01
LON;DUMP COMMON;                                                         SMP02
DUMP PHRASES,LEVEL1;                                                     SMP03
ADD PHRASE:SAMPLE TEST,(1)++FC'PLAN SYSTEM IS OPERATIONAL',LEVEL1;       SMP04
SAMPLE TEST;                                                             SMP05
DELETE PHRASE:SAMPLE TEST;                                               SMP06
```

Figure 1. Sample Problem Listing

```
PLAN JOB,ERASABLE200,MANAGED510,LONG;                                    SMP01
LON;DUMP COMMON;                                                         SMP02
```

Figure 2. Sample Problem Output - Step 2

```
SWITCHES
0000 0000  0000 0000  0000 0000  0000 0000  0000 0000  0000 0000  0000 0000  0000 00C8
0000 047E  0000 01FE  0000 0000  0000 0064  0000 0001  0000 0000  0000 0000

MANAGED ARRAY  LENGTH  510

  1 7FFF FFFF  7FFF FFFF  7FFF FFFF  7FFF FFFF  7FFF FFFF  7FFF FFFF  7FFF FFFF 7FFF FFFF

193 7FFF FFFF  7FFF FFFF  7FFF FFFF  7FFF FFFF  7FFF FFFF  7FFF FFFF  7FFF FFFF 0000 0000
201 0000 000D  D4C1 D5C1  C7C5 C440  C1D9 D9C1  E840 4040  0000 0010  D5D6 D5D4 C1D5 C1C7
209 C5C4 40C1  D9D9 C1E8  E2E6 C9E3  C3C8 C5E2  D3C5 D5C7  E3C8 4040  0000 0064 7FFF FFFF
217 7FFF FFFF  7FFF FFFF  7FFF FFFF  7FFF FFFF  7FFF FFFF  7FFF FFFF  7FFF FFFF 7FFF FFFF

505 7FFF FFFF  7FFF FFFF  7FFF FFFF  7FFF FFFF  7FFF FFFF  7FFF FFFF

NONMANAGED ARRAY LENGTH   0


DUMP PHRASES,LEVEL1;                                                      SMP03
```

Figure 3. Sample Problem Output - Step 3

```
CHECKSUM    1
   PHRASE NAME LIS LIT     LEVEL 1 TYPE-OBJECT    ENTRY SIZE  20   2110    0    0

CHECKSUM    3
   PHRASE NAME OUT         LEVEL 1 TYPE-OBJECT    ENTRY SIZE   8   2346    0    0

CHECKSUM   11
   PHRASE NAME DUM MAN     LEVEL   TYPE-OBJECT    ENTRY SIZE  22   1920    0    0

CHECKSUM   19
   PHRASE NAME DUM PER     LEVEL   TYPE-OBJECT    ENTRY SIZE  28   1764    0    0
CHECKSUM   25
   PHRASE NAME DUM DYN     LEVEL   TYPE-OBJECT    ENTRY SIZE  28   1820    0    0

CHECKSUM   32
   PHRASE NAME DUM COM     LEVEL   TYPE-OBJECT    ENTRY SIZE  22   1876    0    0

CHECKSUM   51
   PHRASE NAME DUM ERR     LEVEL   TYPE-OBJECT    ENTRY SIZE   4   2380    0    0
```

```
CHECKSUM   52
   PHRASE NAME SET LIT     LEVEL   TYPE-OBJECT    ENTRY SIZE  18    2074     0    0


CHECKSUM   72
   PHRASE NAME ALT PHR     LEVEL 0 TYPE-OBJECT    ENTRY SIZE   6    1676     0    0

CHECKSUM  105
   PHRASE NAME DUM PHR     LEVEL 1 TYPE-OBJECT    ENTRY SIZE  65    2150    42    6
   PHRASE NAME CRE COR DIR LEVEL   TYPE-OBJECT    ENTRY SIZE  4     2388     0    0

CHECKSUM  113
   PHRASE NAME PLA JOB     LEVEL 0 TYPE-OBJECT    ENTRY SIZE  31    1698     0    0

CHECKSUM  121
   PHRASE NAME DUM NON     LEVEL   TYPE-OBJECT    ENTRY SIZE  18    1964     0    0

CHECKSUM  149
   PHRASE NAME ADD PHR     LEVEL 0 TYPE-OBJECT    ENTRY SIZE   5    1666     0    0

CHECKSUM  155
   PHRASE NAME CRE LOA ENT LEVEL   TYPE-OBJECT    ENTRY SIZE   4    2396     0    0

CHECKSUM  165
   PHRASE NAME INP         LEVEL 1 TYPE-OBJECT    ENTRY SIZE   7    2332     0    0

CHECKSUM  167
   PHRASE NAME EXE         LEVEL   TYPE-OBJECT    ENTRY SIZE  12    2050     0    0

CHECKSUM  168
   PHRASE NAME DEL PHR     LEVEL 0 TYPE-OBJECT    ENTRY SIZE   5    1688     0    0

CHECKSUM  177
   PHRASE NAME DUM SWI     LEVEL   TYPE-OBJECT    ENTRY SIZE  14    2000     0    0

CHECKSUM  205
   PHRASE NAME LON         LEVEL 1 TYPE-OBJECT    ENTRY SIZE   2    1760     0    0

CHECKSUM  228
   PHRASE NAME CON DUM PHR LEVEL   TYPE-OBJECT    ENTRY SIZE  26    2280     0    0

CHECKSUM  231
   PHRASE NAME SEN         LEVEL   TYPE-OBJECT    ENTRY SIZE   2    2046     0    0

CHECKSUM  239
   PHRASE NAME SAV         LEVEL   TYPE-OBJECT    ENTRY SIZE   9    2028     0    0

CHECKSUM  251
   PHRASE NAME SET PAG LEN LEVEL   TYPE-OBJECT    ENTRY SIZE   9    2362     0    0

END OF PHRASE TABLE DUMP
```

Figure 4. Sample Problem Output - Step 4

```
ADD PHRASE:SAMPLE TEST,(1)+*FC'PLAN SYSTEM IS OPERATIONAL',LEVEL1;        SMP04
SAMPLE TEST;                                                              SMP05
```

Figure 5. Sample Problem Output - Step 5

```
DFJ001    001-100  SAMPLE TEST;
DFJ299 C 00001 SEQ=002 ID=SMP05 PGM=DFJPSCAN PLAN SYSTEM IS OPERATIONAL
DELETE PHRASE:SAMPLE TEST;                                                  SMP06
```

Figure 6. Sample Problem Output - Step 6

APPENDIX B: LISTING OF STANDARD PHRASE FILE

```
ADD PHRASE: ALTER PHRASE,I(1)-1,I(-13)1,PRO'DFJPHRAS,DFJPHRAS',LEVEL0;      ALPH1
ALTER PHRASE: ALTER PHRASE,I(1)-1,I(-13)1,PRO'DFJPHRAS,DFJPHRAS',LEVEL0;    ALPH1
ALTER PHRASE: DELETE PHRASE,I(1)-1,I(-13)1,PRO'DFJPHRAS',LEVEL0;            DEPH1
ALTER PHRASE:PLAN JOB,LEVEL0,I(-1)FILE,SAVED,TO,LISTS,LB,LC,LD,ERASE,       PLJ01
COMMON,MANAGED,NERM,DEVICE,I(1)SHORT-,LONG-,STACK-,IMMEDIATE-,DRIVE0,       PLJ02
DFI-,PFI-,(-11)UMOD,I(-13)FORM0,                                            PLJ03
$0FORM:(LONG)?=FORM+1,FORM:(IMM)?=FORM+2,FORM:(DFI)?=FORM+4,FORM:(PFI)      PLJ04
?=FORM+8,TO=TO+DRIVE*2048;                                                  PLJ05
ALTER PHRASE:LON,LEVEL1;                                                    LON01
ALTER PHRASE:DUMP PERMANENT,I(-8)M,I(M)FILE255,I(M+2)START0,I(M+3)END0,     DUPM1
I(M+4)DRIVE0, (M+5)A'DRIVE    FILE       LENGTH",(M+12)NAME'       ',       DUPM2
I(M+15)NOD100,0,PROGRAM'DFJPFDMP';                                          DUPM3
ALTER PHRASE:DUMP DYNAMIC   ,I(-8)M,I(M)FILE255,I(M+2)START0,I(M+3)END0,    DUPL1
I(M+4)DRIVE0, (M+5)A"DRIVE    FILE       LENGTH",(M+12)NAME'       ',       DUPL2
I(M+15)NOD100,1,PROGRAM'DFJPFDMP';                                          DUPL3
ALTER PHRASE:DUMP COMMON,I(-8)M,I(M)NNN0,'MANAGED ARRAY','NONMANAGED ARRAY'DUMP1
,"SWITCHES","LENGTH",I(M+15)NOD100,PRO'DFJPCDMP';                           DUMP2
ALTER PHRASE:DUMP MANAGED,I(-8)M,I(M)NNN-1,'MANAGED ARRAY',                 DUMM1
'NONMANAGED ARRAY',"SWITCHES","LENGTH",I(M+15)NOD100,PRO'DFJPCDMP';         DUMM2
ALTER PHRASE:DUMP NONMANAGED,I(-8)M,I(M)NNN1,(M+6)B'NONMANAGED ARRAY',      DUMN1
"SWITCHES","LENGTH",I(M+15)NOD100,PRO'DFJPCDMP';                            DUMN2
ALTER PHRASE:DUMP SWITCHES,I(-8)M,I(M)NNN-2,(M+11)A"SWITCHES",              DUMS1
"LENGTH",I(M+15)NOD100,PROGRAM'DFJPCDMP';                                   DUMS2
ALTER PHRASE:SAVE,I(-1)SW,-1,I(-8)M,I(M)FILE0,I(M+1)DRI-1,                  SAVE1
   $0SW:(FIL>0)?=FIL, SW(3):(DRI>-1)&(DRI<5)?=DRI*2048;                     SAVE2
ALTER PHRASE:SEND;                                                          SEND1
ALTER PHRASE: EXECUTE, I(-1)SW,0, I(-8)M, I(M)FROM 0, I(M+1)TO 0,           EXEC1
   I(M+2)FILE 0, I(M+3)DRIVE -1,(M)F*TA'INVALID STATEMENT NUMBER OR DRIVE',EXEC2
$0 SW:(FIL>0)=FIL,  DRI:(DRI<0)?=SW(3)/2028-.5 !:$5, DRI:(DRI<0)?=0,        EXEC3
$5 FRO:¬((DRI>-1)&(DRI<5)) ?=+, SW(3):(TO>0) ?=DRI*2048+TO !=DRI*2048,      EXEC4
SW(2):(FRO>0)?=FRO, FRO:(SW(2)>0);                                          EXEC5
ALTER PHRASE:SET LITERAL,PROG'DFJPDIAG',I(-8)M,I(M)FILE254,I(M+1)NAME0,     SETL1
I(M+4)DRIVE0,I(M+5)NUMBER-*RA'UNDEFINED LITERAL NUMBER',I(M+6)LITERAL0;     SETL2
ALTER PHRASE:LIST LITERALS,LEVEL1,PROGRAM'DFJPLITL',I(1)FILE254,NAME-*A     LISL1
'LITERAL FILE NAME NOT DEFINED',I(5)DRIVE0,NOD100,(35)                      LISL2
"NUMBER  LENGTH    TEXT OF PLAN LITERAL";                                   LISL3
ALTER PHRASE:DUMP PHRASES,I(500)SYS360 ,I(501)NOD100,I(503)LEVEL1,LEVEL1,   DUPH1
              (200)"CHECKSUM","PHRASE NAME","LEVEL     TYPE-OBJECT",        DUPH2
"ENTRY SIZE","VERB","SUBSCRIPT NAME  VALUE     RANGE  INDEX","EXIT PROGRAM DUPH3
 LIST                      ","SYMBOL EXIT FORMAT  SCALE SUBSCRIPT EXPRESSIODUPH4
N","PROGRAM LIST","TEST LOCATION  ACTION","LITERAL, LIST, OR SUBSCRIPT   Y"DUPH5
,"LOCATION  MODE FACTOR EXPRESSION",(510)-*TP'CON DUM PHR',I(504)DRI0;      DUPH6
ALTER PHRASE:CONTINUE DUMP PHRASES,(281)"INTERPRETIVE EXPRESSIONS",         CDPH1
"VERB PROGRAMS","END OF PHRASE TABLE DUMP",PROGRAM'DFJPTDMP',               CDPH2
(505)"DFJPFIL",(835)NAM"DFJPTDP1DFJPTDP2DFJPTDP3DFJPTDP5DFJPTDP6";          CDPH3
ALTER PHRAS: INPUT,I(-8)M,I(M)NOD1,0,LEVEL1,PROGRAM'DFJPIOCS';              INPU1
ALTER PHRAS:OUTPUT,I(-8)M,I(M)A0,I(M+1)NOD101,LEVEL1,PROGRAM'DFJPIOCS';     OUTP1
ALTER PHRAS:SET PAGE LENGTH,I(-8)M,I(M)PGL60,I(M+1)NOD100,PRO'DFJPLENG';    SEPA1
ALTER PHRAS:DUMP ERRORS,PROGRAM'DFJPEDMP';                                  DUER1
ALTER PHRASE: CREATE CORE DIRECTORY, PROGRAM'DFJCRDIR';                     CRC01
ALTER PHRASE: CREATE LOADER ENTRIES, PROGRAM'DFJLLIST';                     CRL01
```

NAME	LENGTH	FUNCTION
DFJCRDIR	1BF8	CORE DIRECTORY BUILD
DFJGMERG	1090	PERMANENT FILE MERGE
DFJGSRTA	1108	PERMANENT FILE SORT
DFJGSRTB	1108	PERMANENT FILE SORT
DFJLLIST	1C50	JOBPAC AREA BUILD
DFJLODER	0CA8	PROGRAM LOADER, SIOCS, DIOCS
DFJPCDMP	1D18	COMMUNICATION ARRAY DUMP
DFJPDIAG	1F00	LITERAL FILE PROCESSOR
DFJPEDMP	12B8	ERROR FILE DUMP
DFJPERRS	3A08	ERROR PROCESSOR
DFJPFDMP	2D68	FILE DUMP UTILITY
DFJPHRAS	4390	PHRASE PROCESSOR
DFJPIDMP	18C0	PHRASE LIST
DFJPIOCS	13C8	UTILITY MODULE
DFJPLAN	3000	MAINLINE EXECUTIVE
DFJPLENG	0FC0	UTILITY MODULE
DFJPLITL	1B10	LITERAL FILE PROCESSOR
DFJPMERG	1AA0	DYNAMIC FILE MERGE
DFJPSCAN	4318	COMMAND INTERPRETER
DFJPSRTA	1B18	DYNAMIC FILE SORT
DFJPSRTB	1B18	DYNAMIC FILE SORT
DFJPSTSV	3270	STATEMENT SAVE
DFJPTDMP	2EC0	PHRASE TABLE DUMP
DFJPTDP1	23F8	PHRASE TABLE DUMP
DFJPTDP2	24F0	PHRASE TABLE DUMP
DFJPTDP3	24F8	PHRASE TABLE DUMP
DFJPTDP5	2730	PHRASE TABLE DUMP
DFJPTDP6	2610	PHRASE TABLE DUMP
DFJRETN	00E8	EOJ PROCESSOR
DFJTRACE	0180	TRACE FACILITY
IGG019WY	0040	SIO APPENDAGE

APPENDIX D: MEMBER LISTING OF PLAN.SUBLIB

NAME	LENGTH	FUNCTION	NAME	LENGTH	FUNCTION
BREAK	0028	CHARACTER MANIPULATION			
CIOEN	0080	INTERNAL CONTROL	PDBFE	0008	SEQUENTIAL FILE SUPPORT
CIOEP	0080	INTERNAL CONTROL	PENDF	00E0	SEQUENTIAL FILE SUPPORT
DFJCGET	0058	INTERNAL CONTROL	PEOF	0050	SEQUENTIAL FILE SUPPORT
DFJDSLL	0110	INTERNAL CONTROL	PEOUT	0058	SEQUENTIAL FILE SUPPORT
DFJPEOUT	*DFJPFOUT	INTERNAL CONTROL	PFIN	0058	SEQUENTIAL FILE SUPPORT
DFJPFIN	01C0	INTERNAL CONTROL	PFND1	*FIND	DYNAMIC FILE SUPPORT
DFJPFOUT	02A0	INTERNAL CONTROL	PFOUT	0058	SEQUENTIAL FILE SUPPORT
DFJPIIN	00C0	INTERNAL CONTROL	PFSPC	00C8	DYNAMIC FILE SUPPORT
DFJPIOUT	00B8	INTERNAL CONTROL	PHIN	02C8	LITERAL PROCESSING
DFJUMC	*DFJUNC	FREE STORAGE CONTROL	PHOUT	08C8	LITERAL PROCESSING
DFJUNC	0248	FREE STORAGE CONTROL	PHTOE	0048	CHARACTER MANIPULATION
ERLST	0010	ERROR FILE DUMP	PIDMP	0118	SPECIAL PSCAN SUBROUTIN
ERRAT	*ERRET	ERROR INTERFACE	PIIN	0058	SEQUENTIAL FILE SUPPORT
ERRET	0178	ERROR INTERFACE	PIOC	0040	SEQUENTIAL FILE SUPPORT
ERREX	*ERRET	ERROR INTERFACE	PIOUT	0058	SEQUENTIAL FILE SUPPORT
ERROR	*ERRET	ERROR INTERFACE	PLINP	0020	SEQUENTIAL FILE SUPPORT
EUSER	*NUSER	USER EXIT	PLOUT	0020	SEQUENTIAL FILE SUPPORT
EWRIT	0188	ERROR FILE PROCESSING	PMERG	00E8	DYNAMIC FILE SUPPORT
FALSE	*TRUE	LOGICAL TEST	PPACK	0020	CHARACTER MANIPULATION
FIND	BA8	DYNAMIC FILE SUPPORT	PPAGL	0038	SEQUENTIAL FILE SUPPORT
FINDL	*FIND	DYNAMIC FILE SUPPORT	PRED1	*FIND	DYNAMIC FILE SUPPORT
GDATA	00B0	PERMANENT FILE SUPPORT	PREL1	*FIND	DYNAMIC FILE SUPPORT
GDAT1	*GDATA	PERMANENT FILE SUPPORT	PSBFA	0008	SEQUENTIAL FILE SUPPORT
GMERG	00E8	PERMANENT FILE SUPPORT	PSBFB	0008	SEQUENTIAL FILE SUPPORT
GSORT	00C0	PERMANENT FILE SUPPORT	PSBFC	0008	SEQUENTIAL FILE SUPPORT
GTVAL	*STVAL	ARRAY TRANSMISSION	PSBFD	0008	SEQUENTIAL FILE SUPPORT
GUSER	*NUSER	USER EXIT	PSBFE	0008	SEQUENTIAL FILE SUPPORT
INPUT	00A0	PHRASE PROCESSING	PSORT	00C0	DYNAMIC FILE SUPPORT
IOCS	0048	INPUT/OUTPUT CONTROL	PUNPK	0020	CHARACTER MANIPULATION
IUSER	*NUSER	USER EXIT	PUSH	0058	PHRASE PROCESSING
LCHEX	00B8	LOADER INTERFACE	PWRT1	*FIND	DYNAMIC FILE SUPPORT
LEX	0018	LOADER INTERFACE	RDATA	0118	PERMANENT FILE SUPPORT
LIST	00B8	LOADER INTERFACE	RDAT1	*RDATA	PERMANENT FILE SUPPORT
LISTB	0050	LOADER INTERFACE	READ	*FIND	DYNAMIC FILE SUPPORT
LNRET	0020	LOADER INTERFACE	RELES	*FIND	DYNAMIC FILE SUPPORT
LOCAL	0030	LOADER INTERFACE	STVAL	0058	ARRAY TRANSMISSION
LREPT	0010	LOADER INTERFACE	TRUE	0020	LOGICAL TEST
LRET	0010	LOADER INTERFACE	WDATA	*RDATA	PERMANENT FILE SUPPORT
LRLD	*LSAV	LOADER INTERFACE	WDAT1	*RDATA	PERMANENT FILE SUPPORT
LSAV	0018	LOADER INTERFACE	WRITE	*FIND	DYNAMIC FILE SUPPORT
NDEF	0038	LOGICAL TEST	XACES	01F8	PHRASE TABLE DUMP
NUSER	0108	USER EXIT	XBIT	00D8	PHRASE TABLE DUMP
PAIN	0098	SEQUENTIAL FILE SUPPORT	XPRNT	0100	PHRASE TABLE DUMP
PAOUT	*PAIN	SEQUENTIAL FILE SUPPORT	XTRAC	02F0	PHRASE TABLE DUMP
PARGI	*PARGO	ARRAY TRANSMISSION			
PARGO	0078	ARRAY TRANSMISSION			
PBFTR	0070	SEQUENTIAL FILE SUPPORT			
PBTST	00B0	BIT MANIPULATION			
PBUSY	0008	SEQUENTIAL FILE SUPPORT			
PCCTL	0060	SEQUENTIAL FILE SUPPORT			
PCOMP	0050	LOGICAL TEST			
PDBFA	0008	SEQUENTIAL FILE SUPPORT			
PDBFB	0008	SEQUENTIAL FILE SUPPORT			
PDBFC	0008	SEQUENTIAL FILE SUPPORT			
PDBFD	0008	SEQUENTIAL FILE SUPPORT			

APPENDIX E: MEMBER LISTING OF PLAN.MACLIB

NAME	CARDS	FUNCTION
CENTR	16	ENTRY MACRO
DFJID	11	TITLE GENERATION
DPLAN	248	LOADER DEFINITION MACRO
EPLAN	20	LOADER DEFINITION MACRO
RPLAN	12	REGISTER EQUATE

YOUR COMMENTS PLEASE...

Your comments on the other side of this form will help us improve future editions of this publication. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material.

Please note that requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or the IBM branch office serving your locality.

fold fold

FIRST CLASS
PERMIT NO. 1359
WHITE PLAINS, N. Y.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY...

IBM Corporation
112 East Post Road
White Plains, N. Y. 10601

Attention: Technical Publications

fold fold

APPENDIX E: MEMBER LISTING OF PLAN.MACLIB

NAME	CARDS	FUNCTION
CENTR	16	ENTRY MACRO
DFJID	11	TITLE GENERATION
DPLAN	248	LOADER DEFINITION MACRO
EPLAN	20	LOADER DEFINITION MACRO
RPLAN	12	REGISTER EQUATE

S/360 PLAN (OS) Operations Manual Printed in U.S.A. GH20-0596-1

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N. Y. 10601
(USA Only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)