

# Publications Bulletin

2200.004-0001  
4/2/73

Release 2 Update Package for:

MRX/OS Control Language Services  
Extended Reference Manual  
2200.004

This bulletin advises of changes that have occurred to the **Control Language Services, Extended Reference Manual** since the **December 1972** edition was issued. New and replacement pages are provided where required.

<u>Pages</u>	<u>Action</u>
v thru viii	Replace
1-3 and 1-4	Replace
1-4a	Add
1-5 thru 1-8	Replace
1-8a	Add
1-11 and 1-12	Replace
1-12a	Add
1-13 and 1-14	Replace
Section 2	Replace
Appendix A	Replace
Appendix B	Replace
Appendix C	Replace
Appendix D	Replace
E-17 and E-18	Replace
Appendix F	Replace
Index	Replace

*Technical* changes to text, tables, and figures are marked with a vertical bar in the outer margin.

Pages containing *non-technical* changes (page layout, spelling corrections) are indicated by a bar opposite the page number.

Please file this bulletin with the publication to retain a record of changes.

Sequence Number: M104

MEMMOREX

Computer System  
Products

# TABLE OF CONTENTS

Section		Page
1	INTRODUCTION	1-1
	Definitions	1-1
	Job and Job Step	1-1
	Control Language Statements	1-1
	Job Queuing	1-3
	Input Data Spooling	1-3
	Output Data Spooling	1-3
	Program Description	1-4
	Input Reader	1-6
	Job Initiator	1-8
	Step Initiator	1-8
	Problem Program	1-11
	Step Terminator	1-11
	Job Terminator	1-11
	Control Flow	1-15
2	CONTROL LANGUAGE	2-1
	Language Description	2-1
	Format	2-2
	Identifier Field	2-3
	Command Field	2-4
	Keyword-Operand Field	2-5
	Comment Field	2-6
	Sequence Field	2-6
	Format Notation	2-7
	Statement Specifications	2-8
	Job Level Statements	2-8
	JOB Statement	2-8
	Sample //JOB Statement	2-10
	//EOJ Statement	2-10
	COMMENT Statement	2-10
	Step Level Statements	2-11
	EXECUTE Statement	2-11
	Sample Execute Statement	2-13
	PAR Statement	2-14
	Sample //PAR Statements	2-14
	DEFINE Statement	2-15
	Scratch Files	2-17
	Temporary Files	2-17
	Work Files	2-17
	Permanent Files	2-18
	Sample //DEFINE Statements	2-21

## TABLE OF CONTENTS (Continued)

Section	Page
Disc Allocation Keywords	2-22
Sample //DEFINE Statement for Disc Space Allocation	2-26
Disc File Expansion Keyword	2-26
Sample //DEFINE Statement for Expansion	2-26
Summary of //DEFINE Statement Keywords	2-27
ROUTE Statement	2-27
Allocation and Expansion Keywords	2-28b
Spooling Information Keywords	2-28c
SET Statement	2-28d
Sample //SET Statements	2-29
TELL Statement	2-30
Sample //TELL Statements	2-31
Interstep Level Statements	2-31
IF Statement	2-31
Sample //IF Statement	2-32
Procedure-Oriented Statements	2-33
DECLARE Statement	2-33
CALL Statement	2-34
Required Run-Time Variables	2-35
Default Values to be Overridden	2-35
Sample //CALL Statements	2-35
Data Level Statements	2-36
DATA Statement	2-37
Sample //DATA Statements	2-40
Data Delimiter Statement	2-41
Job Stream Conventions	2-41
3    FILES AND DEVICES	3-1
General Description	3-1
Special Files	3-1
SYSIN (System Input File)	3-1
SYSOUT (System Output File)	3-2
\$LODLIB (Private Load Library)	3-2
Checkout (Checkout Debugging Directives)	3-2
Device Assignment and File Definition	3-3
Unit Record Devices	3-3
Magnetic Tape Devices	3-3
Standard Labeled Tapes	3-4
Input	3-4
Output	3-4
Non-Standard Labeled Tapes	3-4
Input	3-4
Output	3-5
Unlabeled Tapes	3-5
Input	3-5
Output	3-5

## TABLE OF CONTENTS (Continued)

Section	Page
3 (cont) Telecommunication Devices	3-5
Direct Access Storage Devices (Disc)	3-5
Shared Drive	3-6
Unshared Drive	3-6
Files	3-6
Disc File Organization	3-6
Disc Space Allocation	3-6
Disc File Expansion	3-7
Logical Input/Output	3-7
Physical Input/Output	3-8
4 CATALOGED PROCEDURES	4-1
General Description	4-1
Writing a Cataloged Procedure	4-1
Cataloging a Procedure	4-3
Using Cataloged Procedures	4-3
APPENDIX A – SUMMARY OF CONTROL LANGUAGE STATEMENTS	A-1
APPENDIX B – TABLE OF CONTROL LANGUAGE STATEMENT KEYWORD CHARACTERISTICS	B-1
APPENDIX C – TABLE OF REQUIRED AND OPTIONAL KEYWORDS BY CONTROL LANGUAGE STATEMENT	C-1
APPENDIX D – SYSTEM CONTROL INTERFACE	D-1
APPENDIX E – PARTITION LAYOUT AND USAGE	E-1
APPENDIX F – INDEX – BLOCK SIZE FOR INDEXED FILES	F-1
APPENDIX G – CONTROL LANGUAGE STATEMENTS FOR 2 SAMPLE JOBS	G-1
APPENDIX H – VARIABLE REPLACEMENT RULES	H-1

## LIST OF FIGURES

Figure		Page
1-1	Control Language Services	1-2
1-2	Job Queuing	1-4a
1-3	Input Data Spooling	1-5
1-4	Input Reader	1-7
1-5	Job Initiator	1-9
1-6	Step Initiator	1-10
1-7	Program Execution	1-12a
1-8	Step Terminator	1-13
1-9	Job Terminator	1-14
1-10	Control Language Services Control Flow	1-15
2-1	Indexed File with Spread Factor of 4	2-25
2-2	Sample Job with Card Reader Control by the Program	2-38
2-3	Sample Job Stream	2-41

## LIST OF TABLES

Table		Page
2-1	Sharing of Permanent Files	2-18
2-2	Summary of //DEFINE Statement Keywords	2-28

## JOB QUEUING

Job queuing (Figure 1-2) is a feature of Control Language Services which provides for the entry of jobs from the system card reader into the job queue (reserved area on the disc) to await initiation and processing. A system input file (SYSIN) is created for each job and contains the control language statements for that job. The job which has the highest priority in the job queue, and is allowed to run in the partition, will be initiated first. Priority and partition are specified on the //JOB statement or assumed by default. Where priorities are equal, jobs run on a first-in - first-out basis.

The maximum number of entries in the job queue is specified during system generation (SYSGEN). Whenever more jobs are placed in the card reader hopper than will fit in the job queue, the remainder of the jobs in the hopper wait until interjob time (between jobs), at which point additional entries can be made in the job queue.

## INPUT DATA SPOOLING

Input data spooling, a feature of Control Language Services (Figure 1-3) provides for the creation of sequential disc data files from card images as they are encountered in the card input stream. When input data spooling is performed, jobs which follow the data statement in the card reader are placed in the job queue. If data spooling is not performed, that is, the programmer specifies the system card reader as his input device, any following jobs in the card reader cannot be placed in the job queue until the job with card reader control has completed its processing. That job which selects the system card reader for its input device will be selected for initiation in the same manner as all other jobs in the job queue — by partition and priority.

## OUTPUT DATA SPOOLING

Output spooling is an option that may be selected at SYSGEN time. If the spooling option is selected, the files to be spooled are queued on the output spool queue at step termination, or the files may be queued by the SPOOLQ macro (refer to **MRX/OS Control Program and Data Management Services, Basic Reference** manual). SYSOUT is queued at job termination.

If the spooling option is not selected at SYSGEN time, the //ROUTE statement is treated as if it were a //DEFINE statement; and a physical device (if available) is assigned. No Control Language Services errors result when a //ROUTE statement is encountered in a system that does not have the spooling option, unless an error would have occurred if the option were selected.

The output scheduled for a spooled device is determined by job priority, current form type, and job name. Therefore, print/punch files of a job will not necessarily be output contiguously on the same spooled device. However, continuity can be achieved by specifying a specific device with the same form type for all files.

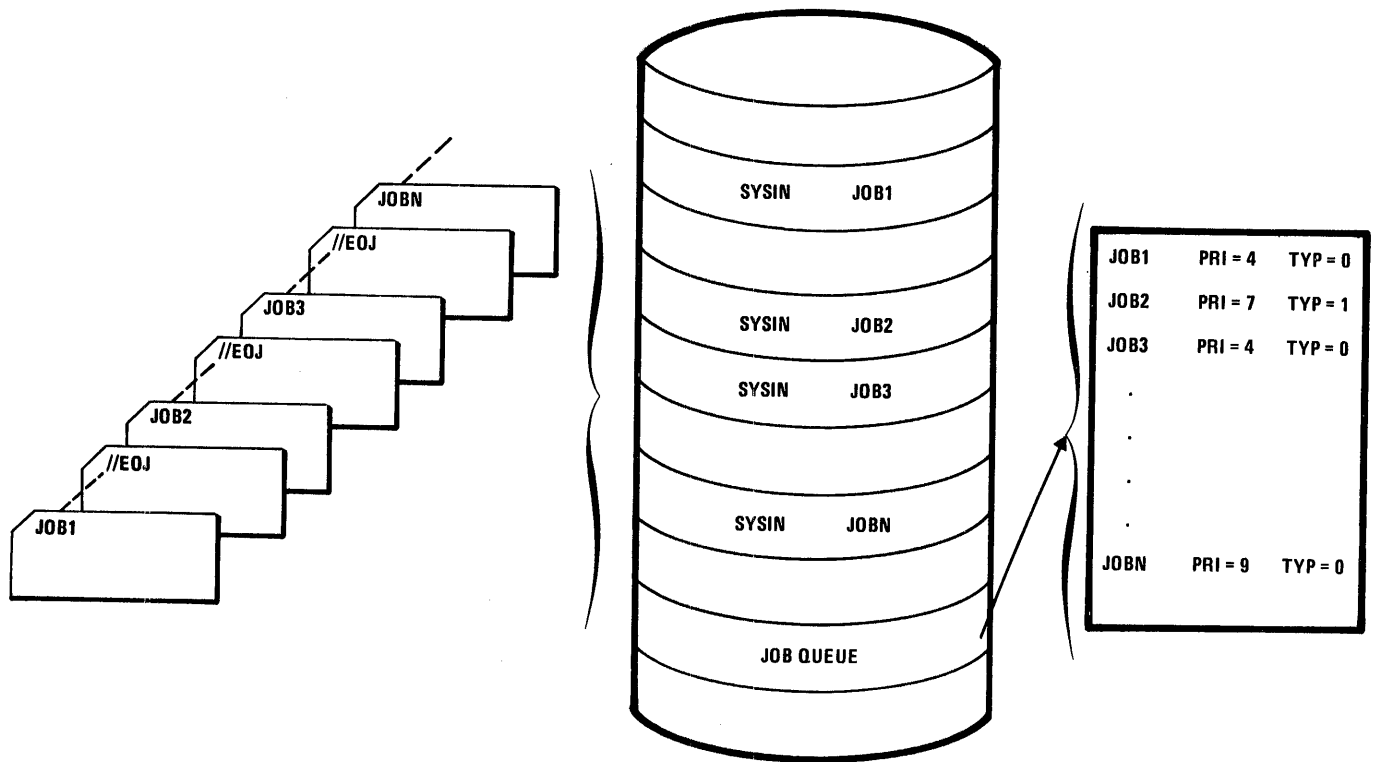
## PROGRAM DESCRIPTION

Control Language Services consists of several modules stored on the system resident pack. Logically, these modules include:

- Input Reader
- Job Initiator
- Step Initiator
- Step Terminator
- Job Terminator

When required, modules are transferred into the problem program partition for execution between jobs and job steps. A minimum main storage partition of 8K bytes, which includes space for necessary control tables, is required for the modules.

In a two-partition system, Control Language Services executes in alternate partitions as the jobs terminate in them, whereas in a one-partition system, jobs are initiated serially.



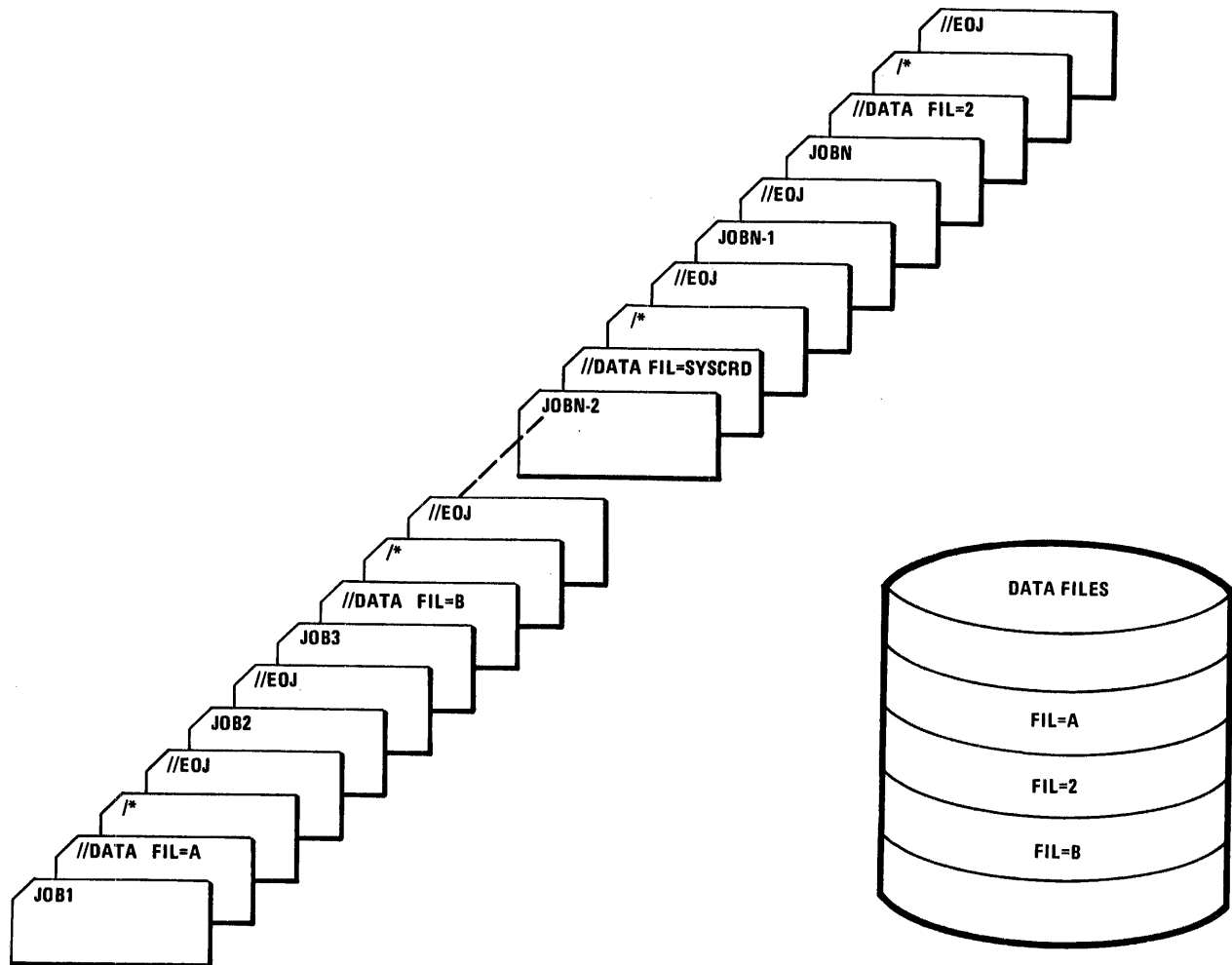
**FUNCTIONS:**

- JOBS ARE LISTED ON THE JOB QUEUE FOR INITIATION AND PROCESSING AS THEY ARE READ IN FROM THE SYSTEM CARD READER
- JOBS ARE THREADED TO THE OTHER JOBS OF THE SAME PRIORITY
- THE FIRST JOB READ AT INITIAL PROGRAM LOAD TIME ALLOWED TO RUN IN THE PARTITION WILL BE INITIATED REGARDLESS OF PRIORITY
- ALL OTHER JOBS ARE INITIATED BY PRIORITY IN THE PARTITION IN WHICH THEY ARE ALLOWED TO RUN
- JOBS OF EQUAL PRIORITY ARE INITIATED ON A FIRST-IN-FIRST-OUT BASIS IN THE PARTITION IN WHICH THEY ARE ALLOWED TO RUN
- JOB1 READ, QUEUED AND INITIATED
- JOB2 READ AND QUEUED
- JOB3 READ AND QUEUED
- JOBN READ AND QUEUED, HIGHEST PRIORITY WILL BE INITIATED BEFORE JOB2 OR JOB3 PROVIDED JOBN IS IN THE QUEUE BEFORE NEXT JOB INITIATION

**NOTE: REPRESENTATION OF JOBS DOES NOT SHOW ACTUAL STATEMENTS**

**Figure 1-2. Job Queuing**





**FUNCTIONS:**

- READS DATA FROM WITHIN JOBS AND WRITES THIS DATA ONTO DISC
- ALLOWS LATER JOBS TO BE QUEUED
- JOB1, JOB3, AND JOB N REQUEST SPOOLING  
JOB2 AND JOB N-1 HAVE NO DATA ENTERED WITH THEM  
JOB N-2 HAS CONTROL OF CARD READ
- JOBS 1, 2, 3, and N-2 WILL BE QUEUED  
FILES A, B SPOOLED  
JOB N-2 GETS CONTROL OF CARD READER UNTIL ITS CARD DATA FILE IS PROCESSED AND THE JOB IS TERMINATED.  
ONLY THEN CAN JOBS N-1 AND N BE QUEUED, AND FILE 2 SPOOLED.

**NOTE: REPRESENTATION OF JOBS DOES NOT SHOW ACTUAL STATEMENTS**

**Figure 1-3. Input Data Spooling**

## INPUT READER

This module of Control Language Services is loaded into the problem program partition at interjob time. As its name implies, the Input Reader reads jobs from the system card reader. Upon reading the first statement of a job (//JOB card), space for the system input (SYSIN) and the system output (SYSOUT) files are allocated. As each control language statement is read, it is output to SYSIN, thus building that file. (Explanation of these files is in Section 3 of this document.) The Input Reader requires a partition of at least 8192 bytes, but does not use more storage even if available.

Cataloged procedures which consist of control language statements cataloged on a library file, are merged with the job, if called. (See Section 4 for a discussion of cataloged procedures.)

A Control Language Services routine called Statement Interpreter is loaded with the Input Reader. The Statement Interpreter performs syntax checking on control statements input from the system card reader as well as those cataloged procedures which may have been merged.

Data files in the job stream are also read by the Input Reader, if data spooling is to be performed.

When all the control statements for a job (including cataloged procedures) have been read and found to be error free, the job is placed on the job queue to wait for initiation. The Input Reader will continue to process jobs until the job queue is filled, the card reader hopper is empty, or a job specifies control of the system card reader. The first job in after autoloading, however, is initiated immediately.

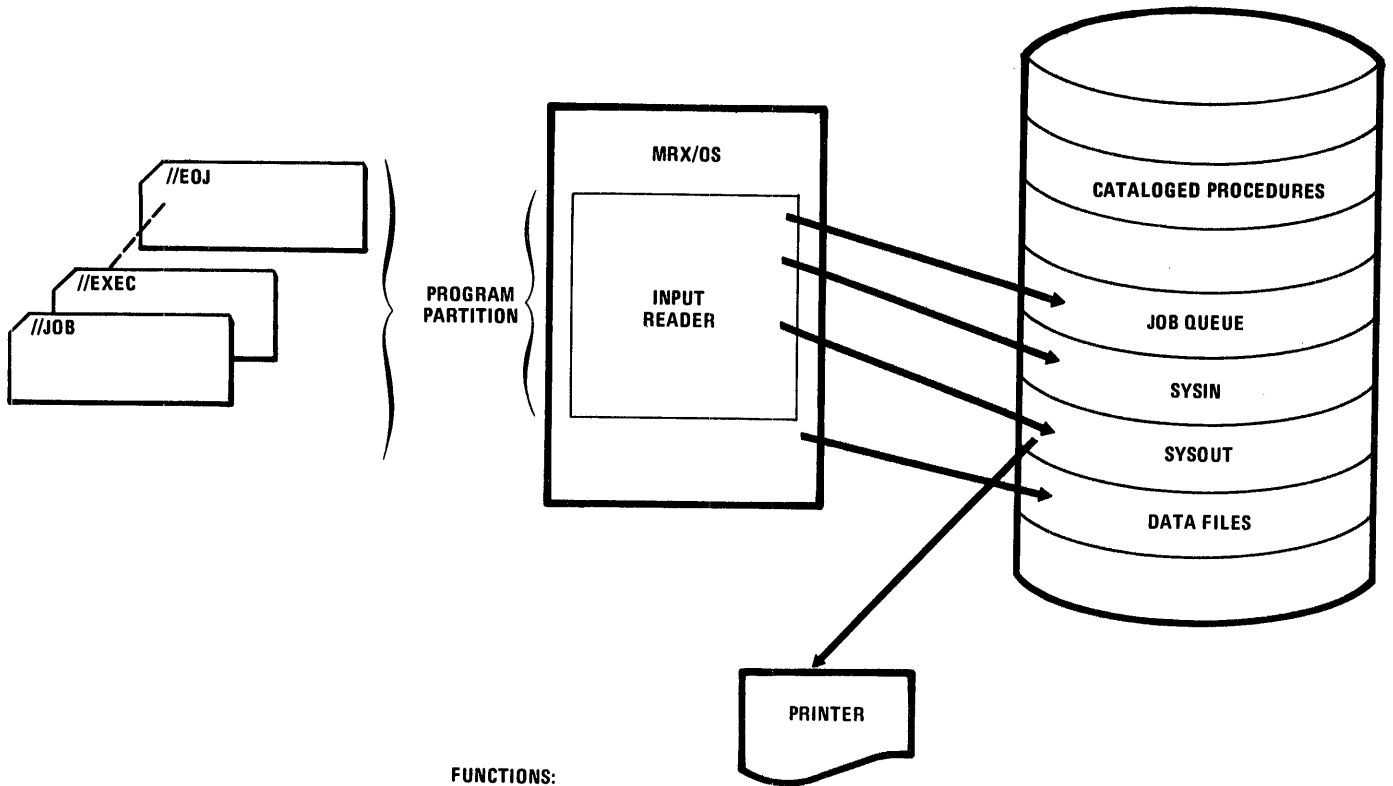
After it has stopped reading jobs from the system card reader, the Input Reader searches the job queue for a job to be initiated in the partition. The Input Reader selects from the entries in the job queue as follows:

1. Jobs assignable to run in the partition.
2. Within those jobs, the jobs of the highest priority.
3. Within those jobs, the first job entered into the job queue (first-in, first-out).

As soon as a job has been selected for initiation, the Input Reader releases control of the partition. A request is made to the Relocating Program Loader to load the Job Initiator.

If syntax errors are detected within a job, the control statements that have been written on SYSIN are transferred to the SYSOUT file. Those statements which contain errors are written with the error indication immediately following the statement. The format of this output is: the statement in error, followed by a flag indicator in the form of a question mark (?), and an error code followed by a description.

```
Example: //DEEF  
         ?  
         nnn UNRECOGNIZABLE COMMAND
```



**FUNCTIONS:**

- READS CONTROL LANGUAGE STATEMENTS
- MERGES CATALOGED PROCEDURES
- BUILDS JOB QUEUE
- ALLOCATES AND BUILDS SYSIN
- ALLOCATES SYSOUT
- READS DATA INPUT STREAM
- BUILDS SPOOLED FILES FROM DATA IF SPOOLING IS REQUIRED
- BUILDS DEBUG FILES IF REQUIRED
- REJECTS JOBS WHICH HAVE CONTROL LANGUAGE ERRORS, BUILDS AND PRINTS SYSOUT FOR REJECTED JOBS
- SELECTS A JOB FOR INITIATION
- CALLS JOB INITIATOR VIA LOADER

**NOTE: REPRESENTATION OF JOBS DOES NOT SHOW ACTUAL STATEMENTS**

**Figure 1-4. Input Reader**

The remainder of the statements for that job are processed and written on the SYSOUT file along with information on detected errors for each statement. The Input Reader then prints these statements from the SYSOUT file up to the last statement containing an error. SYSIN and SYSOUT are closed and purged and the next job is read by Input Reader.

## JOB INITIATOR

Following the selection of a job to be initiated in a partition, the Job Initiator receives control from the Input Reader. Date and switches are initialized. Job Initiator then opens the SYSIN and SYSOUT files for the job, causing file description tables to be created for them. Entries are made for the job in its Job Control Table (JCT), located near the end (high addresses) of the partition.

Jobs are selected for initiation by TYPE and PRIORITY in queued systems. The TYPE designates whether the job is eligible to run in the partition; and of those eligible, the one with the highest priority will be initiated first. When priorities are equal, initiation is on a first in-first out basis.

Job Initiator requires at least 4096 bytes to initiate a job.

The Job Initiator then releases control of the partition and a request is made to the Relocating Program Loader to load the Step Initiator module of Control Language Services.

## STEP INITIATOR

Whenever a job step is to be executed in a partition, the Step Initiator is loaded into the partition and control is passed to it.

The Step Initiator processes all of the statements of a step from an //EXECUTE to the following //EXECUTE, //IF, or //EOJ. Pointers to //DEF and //PAR statements (which define data files and supply parameters) as well as information from control statements, contained on the SYSIN file, are moved to the Job Control Table. The Step Initiator outputs //TELL messages to the operator's console as it encounters them in the job stream and waits for response whenever PAUSE=YES is coded.

For a program requesting restart, a flag is set in the //DEFINE record for this file if the file has been allocated. If an "already exists" error occurs, Step Initiator ignores the error when restart is specified. When RESTART=nnn is coded, the checkpoint number is set in the Job Control Table (JTCCHKP).

In the processing of //DEFINE statements, Step Initiator outputs mount messages for volumes required but not yet mounted; locates files which do not require allocation; selects devices for unit record and tape assignments; and requests disc space through the ALLOCATE routine. (See the **Control Program and Data Management Services, Extended Reference** manual for a description of the ALLOCATE macro.) Usage conflicts for files (requests for conflicting use of non-shared and shared devices) are also cleared.

Once all statements of a step have been processed the BEGIN PGM program-name message is displayed on the console, and the program named in the //EXECUTE statement is loaded and control is passed to that program. When it has completed execution, the Step Terminator is loaded and control is passed to it. If further errors are encountered during step initiation, Step Initiator will cause the job to be aborted and SYSOUT listing to be provided.

After the step has been terminated by the Step Terminator, the Step Initiator again is loaded and receives control to process the statements of the next step. If the next control language statement is an //IF statement, either the next sequential statement is processed or the step named on the //IF statement is processed. The result of a test determines which of these paths will be taken. (The //IF statement is discussed in Section 2 of this document.)

If the next statement is an //EXECUTE, the statement is processed by the Step Initiator as previously described. When the next statement is an //EOJ, a request is made to the Relocating Program Loader to load the Job Terminator module of Control Language Services.

### **PROBLEM PROGRAM**

The problem program is the program named on the //EXECUTE statement. Step Initiator requests the loading of the program into the partition and passes control to this problem program.

The program is responsible for opening and closing the files it will use, and for calling the Relocating Program Loader to load additional segments of the program. When the program has completed execution, its final service request is a HALT, EHALT, or ABEND. HALT is for normal step termination. An EHALT is for job termination. ABEND requests an abnormal termination of a job. These service requests cause the Relocating Program Loader to load the Step Terminator and pass control to that Control Language Services module.

### **STEP TERMINATOR**

Whenever the problem program in a step has completed execution (that is, goes to HALT or EHALT or ABEND), the Step Terminator is loaded. The Step Terminator checks for any open data files. All files opened by the program must be closed by the program at the end of each job step. Failure to do so results in abnormal termination of the step and job (with the exception of SYSCHK).

If the program terminates abnormally and the step is eligible for restart, the operator is asked to allow an immediate restart at the last checkpoint taken by the program. If the operator reply is YES, the Step Terminator sets the immediate restart bit (bit 1 of JTCLS) in the Job Control Table and loads the RESTART program.

All scratch files are purged (de-allocated) by the Step Terminator unless an immediate restart is in progress or abnormal termination is occurring and the user program issued the CHKPT macro. All peripheral devices assigned to the step are released for use by following steps or by the other partition.

The Step Terminator closes the DEBUG file and private load library, if used by the step. Job accounting information is placed on the SYSOUT file and the Step Initiator is loaded.

Step Terminator also searches the SYSIN file for a file to be spooled; and if found, it queues the file on the output writer queue.

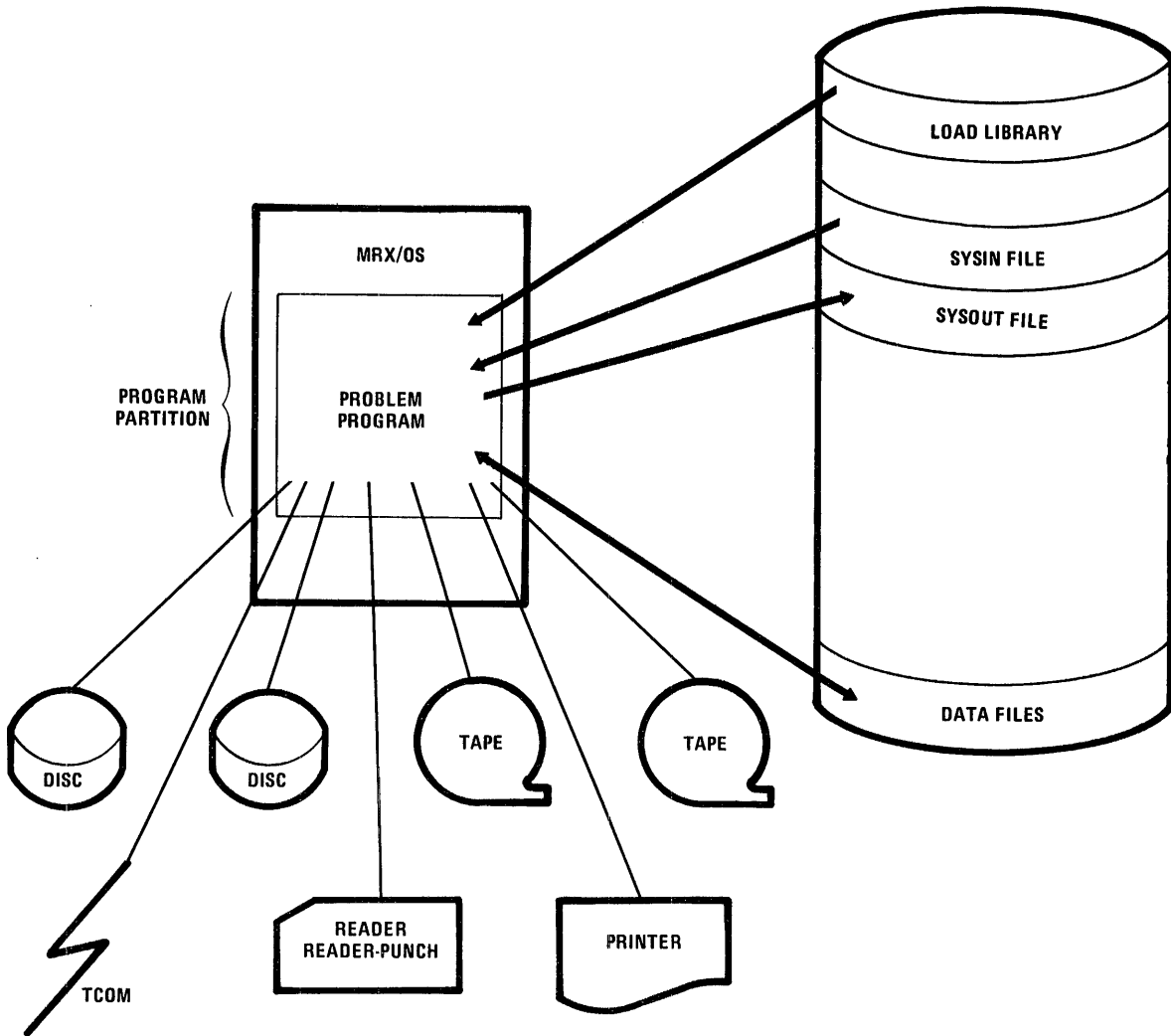
### **JOB TERMINATOR**

Whenever an //EOJ Control Language Statement is read by the Step Initiator, the Job Terminator is loaded into the partition.

Job Terminator closes and purges the SYSIN file. The SYSOUT file is updated with job accounting information and is printed. If termination is normal or is abnormal with no checkpoints taken, user temporary files are purged. If termination is abnormal and checkpoints have been taken, purging of temporary files including SYSIN is bypassed. If spooling was not selected at SYSGEN time, SYSOUT is printed and purged. If spooling was selected, SYSOUT is queued for printing by the output writer.

All other files (work and permanent) remain on the packs until they are de-allocated via the Purge utility or Purge macro. (The Purge utility is described in the **Utility Programs Reference** manual; the Purge macro is described in **Control Program and Data Management Services, Extended Reference** manual.)

When all termination activities have been completed, Job Terminator releases control of the partition. The Input Reader is then loaded into the partition and control is passed to it.

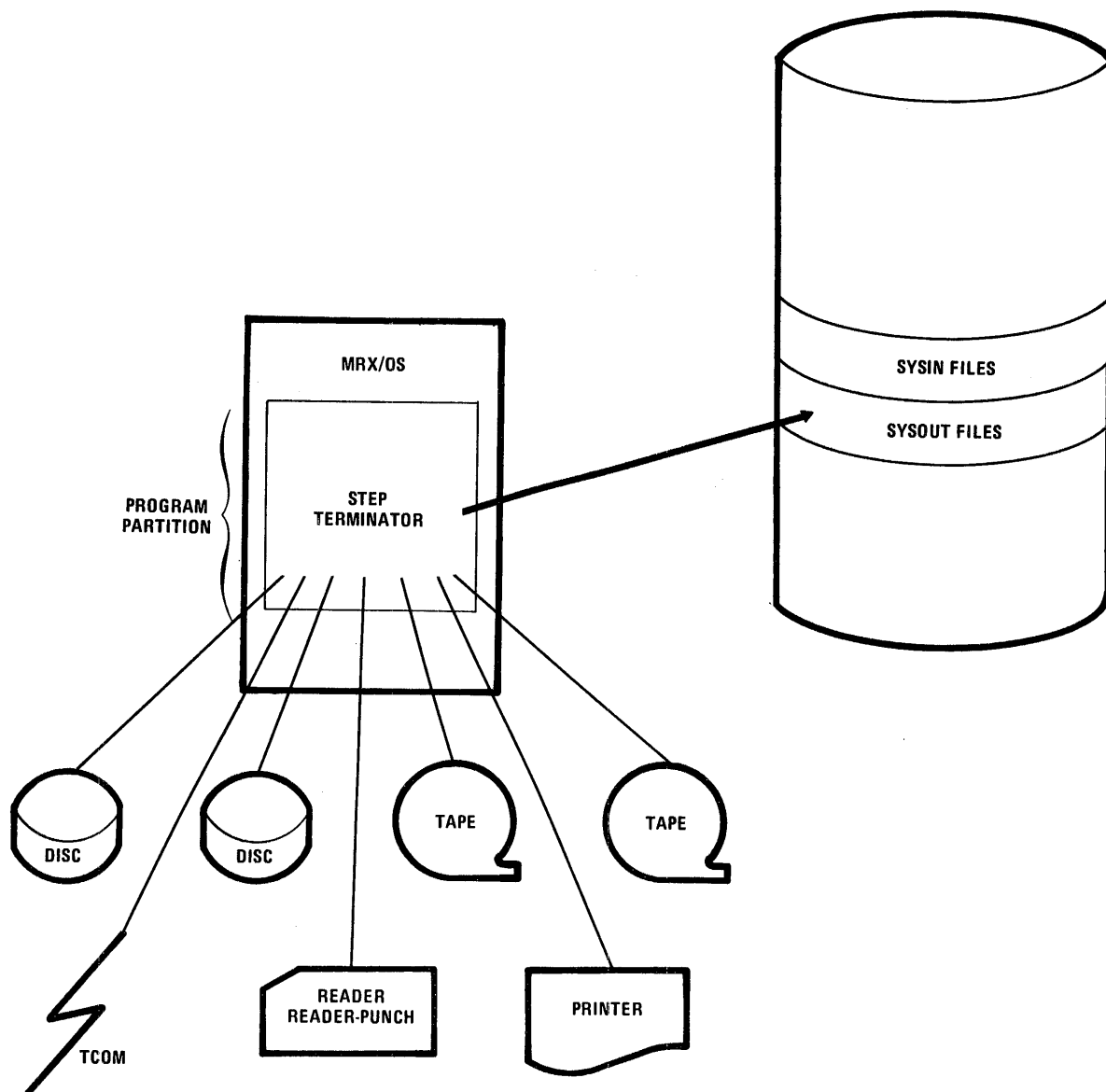


**FUNCTIONS:**

- EXECUTES THE PROBLEM PROGRAM
- ACCEPTS PARAMETER STATEMENTS FROM SYSIN FILE
- DISPLAYS MESSAGES ON SYSOUT FILE
- OPENS AND CLOSES ALL DATA FILES USING INFORMATION SUPPLIED BY DEFINE STATEMENTS
- READS OR WRITES DATA ON ALL ASSIGNED I/O DEVICES
- ALSO READS DATA FROM SPOOLED DATA FILES CREATED BY INPUT READER
- PROGRAMS WITH AN OVERLAY STRUCTURE CALL PROGRAM SEGMENTS FROM LOAD LIBRARY VIA PROGRAM LOADER
- EXECUTES ANY DEBUG INSTRUCTIONS PLACED IN PARTITION
- AT TERMINATION (ABNORMAL OR NORMAL) OF PROGRAM, STEP TERMINATOR IS CALLED VIA THE RELOCATING PROGRAM LOADER

Figure 1-7. Program Execution

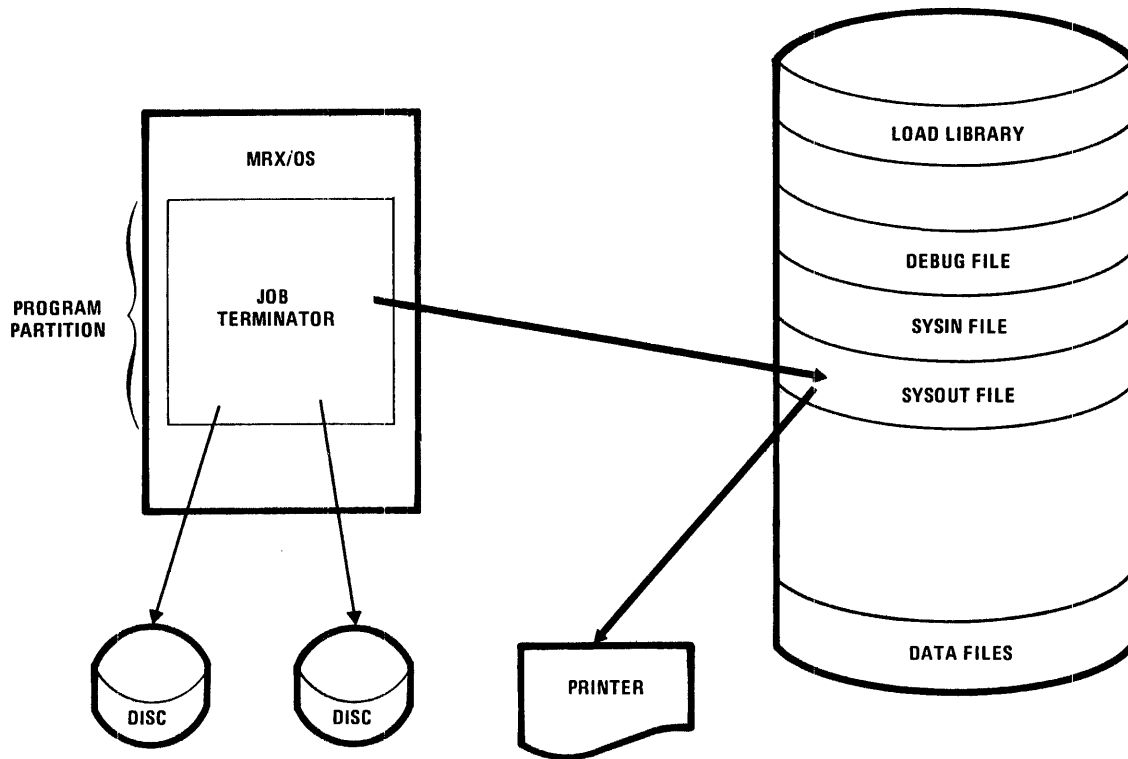




**FUNCTIONS:**

- DEASSIGNS DEVICES WHICH STEP INITIATOR ASSIGNED TO THIS STEP
- PURGES ANY SCRATCH FILES USED BY THIS STEP
- CLOSSES DEBUG FILE AND PRIVATE LOAD LIBRARY, IF USED
- CHECKS FOR ANY FILES OPENED BUT NOT CLOSED BY PROGRAM; IF ANY, ABNORMALLY TERMINATES STEP AND JOB
- PLACES ACCOUNTING INFORMATION ON SYSOUT FILE
- CALLS STEP INITIATOR VIA THE RELOCATING PROGRAM LOADER TO DETERMINE WHETHER A FOLLOWING STEP IS TO BE INITIATED BY STEP INITIATOR, OR THE JOB IS TO BE TERMINATED BY JOB TERMINATOR

**Figure 1-8. Step Terminator**



**FUNCTIONS:**

- CLOSES AND PURGES SYSIN FILE
- WRITES JOB ACCOUNTING INFORMATION TO SYSOUT
- CALLS INPUT READER VIA THE RELOCATING PROGRAM LOADER
- PURGES TEMPORARY FILES FOR NORMAL TERMINATION OR ABNORMAL TERMINATION WITH NO CHECKPOINTS TAKEN
- PRINTS AND PURGES SYSOUT IF SPOOLING OPTION NOT AVAILABLE
- QUEUES SYSOUT FOR PRINTING BY OUTPUT WRITER IF SPOOLING OPTION IS AVAILABLE

Figure 1-9. Job Terminator

## 2. CONTROL LANGUAGE

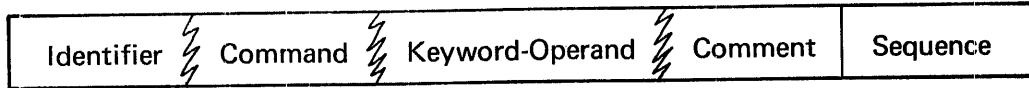
### LANGUAGE DESCRIPTION

The Control Language provides the programmer with a direct interface to Control Language Services. The language is composed of the following statements which control job step, interstep, and procedure-oriented activities for the job; input data to the system; and provide comments.

<u>Level</u>	<u>Statement</u>	<u>Description</u>
JOB	//JOB	Identifies the job.
STEP	//EXECUTE	Specifies the program to be executed.
STEP	//PAR	Specifies information to be passed to the executing program.
STEP	//DEFINE	Specifies physical and logical qualities of a file.
STEP	//ROUTE	Specifies allocation information when spooling option is selected, or specifies physical qualities of unit record device when spooling option not selected.
STEP	//SET	Alters the job date and setting of bits in the program option switch table.
STEP	//TELL	Output message on operator's console.
INTERSTEP	//IF	Tests and transfers control to a subsequent step as a result of the test.
PROCEDURE	//DECLARE	Identifies first statement of cataloged procedure.
	//CALL	Used to merge cataloged control language statements (procedures) into the control language statements of a job.
DATA	//DATA	Identifies that which follows as a data file. Specifies file name and attributes.
DATA	/*	Data delimiter. Identifies the end of a data file.
JOB	//EOJ	Identifies the end of a job.
JOB	*(comment)	Comments card.

## FORMAT

All command statements for the control language have the same basic format containing five fields:



The following general rules apply:

1. Each statement must begin on a new card.
2. The identifier field must begin in character position 1 of each statement.
3. The command field must begin in character position 3.
4. All fields must be separated by at least one blank character, except that no space is allowed between the identifier and command fields.
5. The keyword-operand and comment fields are free form.
6. Keyword-operands are separated by commas. No blank characters are allowed, except on //PAR and //TELL statements.
7. The keyword-operand field may be continued on more than one card except for the //PAR and //TELL statements.
8. A comma followed by a blank indicates that the keyword-operand field is continued on the next card.
9. On continuation cards the keyword-operand field must not begin before character position 4. The identifier field is repeated, but the command field is not.
10. The sequence field occupies the last eight character positions of the card.

## **IDENTIFIER FIELD**

The identifier field indicates that the statement is a directive to the Control Language Services. The identifier is required on all control statements, including all continuations.

The following rules apply to the identifier field:

1. The // identifier must appear in character positions 1 and 2 for all control statements (including continuations) except the data delimiter and the comment statement.
2. The data delimiter identifier is /\* (slash, asterisk, blank) which must appear in character positions 1, 2, and 3. Anything other than a blank in character position 3 will cause the card to be treated as data. Continuation is not applicable for this statement.
3. The comment statement identifier is \* and must appear in character position 1. Continuation is not applicable for this statement.

## COMMAND FIELD

The commands recognized by Control Language Services function is required. The following rules apply to the command field:

1. The commands recognized by Control Language Services must appear in one of the following spelled out or abbreviated forms.

<u>Level</u>	<u>Command</u>	<u>Abbreviation</u>
Job	{ //JOB	None
	{ //EOJ	None
Step	{ //EXECUTE	//EXEC or //EX
	{ //PAR	None
	{ //DEFINE	//DEF
	{ //ROUTE	//RTE
	{ //SET	None
	{ //TELL	//TEL
Interstep	//IF	None
Procedure	{ //DECLARE	//DEC
	{ //CALL	//CAL
Data	//DATA	//DAT

Note that there are no commands for the data delimiter and the comment statement.

2. The command field must begin in character position 3.
3. The command field is not repeated for statement continuations.
4. No space is allowed between the identifier field and the command field.
5. At least one space is required between the command field and any following field.

## KEYWORD-OPERAND FIELD

The keyword-operand field lists the specifications which support the job processing features of the command as required at run time. The following rules apply to the keyword-operand field.

1. The keyword-operand field must be separated from both the preceding and the following fields by at least one blank space.
2. The field is free form with no fixed starting position, although each operand or variable symbol is limited to 17 characters maximum.
3. Each keyword-operand must consist of a descriptive keyword and an operand or variable symbol, separated by an equal sign (except on //PAR cards, which do not undergo syntax check).

Example: LOC=YES

4. Keyword-operands can be separated only by commas (no blanks).

Example: CAT=NO,VOLUME=65AR22,VER=YES

5. If the operand portion of the keyword-operand contains more than one entry, the multiple items are separated by commas, and the entire operand is enclosed by parentheses.

Example: STATUS=(P,O)

6. The keyword-operand field may occupy multiple cards. The continuation of the field must be shown by a comma followed by a blank. The field may not extend past column 72. The only valid characters in column 72 is a blank or a comma.
7. For continuation, the field can only be divided after a complete keyword-operand. The continued line must contain the identifier but must not contain the command.

Example: //DEF ID=FIL128,FIL=PROG6382,  
// NUM=1450,CSD=NO,SIZ=50,  
// BLK=10,LOC=YES,ORG=I,  
// SPREAD=4,STA=(P,O),  
// CAT=NO,VOL=65AR22,  
// VER=YES

8. The number of specifications which may be placed on a record is limited only by the number of character positions not committed to other fields (identifier on all statements, command on the first record of a statement, and sequence field on all statements). Either single or multiple specifications may be included on any record of a statement.

9. The order of the keyword-operands on the statement is optional. There are no positional keyword-operands in the control language except on the //JOB and //CALL statements.

Example: //DEF ID=FIL128,FIL=PROG6382,NUM=1450

This statement could be written with the keyword-operands in any order, such as:

```
//DEF FIL=PROG6382,ID=FIL128,NUM=1450
```

or

```
//DEF NUM=1450,FIL=PROG6382,ID=FIL128
```

### **COMMENT FIELD**

The comment field is used to record additional information concerning the program, the statement, or any documentation the programmer wishes to provide. The field is optional for all statements.

Although there is no specified format for this field, the following rules apply to its general use:

1. The field is free form in that it occupies no specified character positions on the card. Its only restriction is that it must be separated from both the preceding and following fields by at least one blank character.
2. No continuation of comments is allowed.
3. For the comment statement, the comment field may begin in any character position after the \* identifier.

### **SEQUENCE FIELD**

The sequence field is used to indicate the order in which statements are expected to appear in a job. The following rules apply to the sequence field:

1. The field occupies the last eight character positions of each card.
2. The field must be preceded by one or more blank characters.
3. All members of the EBCDIC code set can be used for the values in the sequence field.
4. Control Language Services will check ascending sequence and issue warning diagnostics for exceptions.



## FORMAT NOTATION

In the following paragraphs of this section, each control language statement is discussed. The statement formats are given using all acceptable forms on each command and keyword. The following rules are used in statement formatting in this section:

1. All forms of the command are given. Any form may be used in coding.

Example:     //EXECUTE  
              //EXEC  
              //EX

2. All references to alphanumeric fields, whether they be 8 or 17 characters maximum length, have the following restrictions. Alphabetic and numeric characters comprise the field, no embedded blanks are allowed. The dollar sign (\$) is valid. (Data Management allows the dollar sign only as the first character of a file name; certain system files are preceded with the \$ to identify them as system files.) The dash (-) is the only other special character that may be used within these alphanumeric fields.

Example:     NAME=STEP2-A  
              LIB=\$UTIL-LIBRARY

3. Braces { }, indicate that the user has a choice of terms; the default case is underlined.

Example:     CLS= { YES }  
                  NO }

4. All forms of optional keyword-operands or optional operands are enclosed by brackets [ ].

Example:     [MSC=code]

5. Parentheses ( ) are used to enclose multiple operands associated with one keyword. The parentheses are not required if only one of the operands is specified for use.

Example:     STATUS=(type,usage)

## STATEMENT SPECIFICATIONS

The keyword-operands for control statements are defined by the control language, where each command is associated with specific keywords. The paragraphs which follow describe the keyword-operand specifications for each of the control language statements. The following appendices may be used along with the discussion of each statement.

- Appendix A – Summary of Control Language Statements
- Appendix B – Table of Control Language Statement Keyword Characteristics
- Appendix C – Table of required and Optional Keywords by Control Language Statement

## JOB LEVEL STATEMENTS

The job level statements are used to begin and end the command sequence of a job, and to provide commentary documentation. These statements include:

- //JOB
- //EOJ
- \* (comment statement)

## JOB STATEMENT

```
//JOB      { NAME } =job-name
           { NAM }
           [ { USER } =user-identification ]
           [ { USE }
           [ { TYPE } =partition-number ]
           [ { TYP }
           [ { PRIORITY } =job-priority-number ]
           [ { PRI }
           [ { HOLD } = { YES } ]
           [ { HLD } = { NO } ]
```

The //JOB statement is required to begin the command sequence for a job. This statement describes the job to Control Language Services; assigns a name to the job; identifies the user; specifies the partition in which the job may run; and indicates the priority to be assigned to the job in the job queue. It also specifies whether the job is available for initiation, or is being held pending later release by the operator. If any error is present, the job card is not recognized and Control Language Services skips for job card.

**{ NAME }  
{ NAM } =job-name (required)**

The system requires a name by which it can recognize and refer to a job. This information is supplied by the NAME keyword and is used in creating the job queue entry. In addition, NAME is used to qualify scratch and temporary files; SYSIN and SYSOUT files; and spooled data files.

The operand must be unique (only one job of a name may be in the system at one time). The operand given with the NAME keyword must be a 1-to-8 character alphanumeric value. *This keyword-operand is required and must appear on the first card.*

**{ USER }  
{ USE } =user-identification**

Installations have the option of requiring user identification to the system for job accounting purposes. If this information is not required, the USER keyword is optional; it may be included in the statement merely as added documentation. The choice of requiring user identification is made at system generation (SYSGEN) time. It may be reversed on any following generation of the system.

The operand is a 1-to-4 character alphanumeric field. There is no default.

**{ TYPE }  
{ TYP } =partition-number (optional)**

The partition in which a job is allowed to run may be selected by the programmer, using the TYPE keyword. The keyword may be used to reflect partition size, partition dedication, or job requirements. Partitions are defined at SYSGEN time or at IPL time.

The operands are 1 and 2 specifying partition 1 or 2, respectively. The system default is zero, which means that the job can run in either partition.

**{ PRIORITY }  
{ PRI } =job-priority-number (optional)**

The system provides the ability to designate the order in which jobs run within a partition. The PRIORITY keyword is used to define that order.

Jobs are selected from the queue first by the TYPE keyword (jobs eligible to be initiated in the partition) and then according to the PRIORITY keyword (highest priority job within a partition). Within a partition, jobs of the same priority are selected for initiation on a first-in, first-out basis.

The operand is a 1-digit numeric field of values from 1 to 9. The highest priority is designated by the largest number. A default is set by the installation at SYSGEN time.

$\left\{ \begin{array}{l} \text{HOLD} \\ \text{HLD} \end{array} \right\} = \left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$

(optional)

Control Language Services allows the operator to control the release of a job from the job queue. This control is provided by the HOLD keyword.

When HOLD=YES is encountered, a message is displayed on the operator's console informing the operator that the named job is being held in the job queue. After the operator has released the job, it will be selected for initiation according to the operands of the TYPE and PRIORITY keywords specified. If type and priority are not specified, the installation defaults are used and the job will be selected on a first-in, first-out basis as per those values.

When HOLD=NO is specified, or the HOLD keyword is not specified, the job will be selected for initiation without operator intervention. The same conditions for job selection apply.

#### Sample //JOB Statement

The following is an example of a //JOB statement. The required keyword, NAME, and some optional keyword-operands are used.

```
//JOB NAME=JOB6A2,TYP=2,PRI=8,HOLD=NO
```

In this example, the job name is JOB6A2, which must be unique. The job is allowed to run in partition 2 at a priority of 8. This job will not be held in the job queue for operator release. Because the USER keyword-operand is omitted, it is assumed that the installation does not require user identification. If USER is required, the job will be rejected by the Input Reader Module of Control Language Services.

#### EOJ STATEMENT

```
//EOJ
```

The //EOJ statement is required to terminate the command sequence for a job. There are no keyword-operands associated with this statement, but the comment and sequence fields may be used.

#### COMMENT STATEMENT

```
*comment
```

The comment statement is provided to allow the programmer to include comments in the job stream. It is never required. The comment statement may occur any place within a job.

The comment statement is identified by \* in character position 1; the actual comment may begin in any character position thereafter. Since there are no command and

keyword-operand fields associated with this statement, the comment can occupy the remainder of the card except for the sequence field (last eight character positions).

### Sample Comment Statement

\*OUTPUT IS PROCESSED BY JOB REPORT 62

### STEP LEVEL STATEMENTS

The statements at the step level specify the program to be executed and indicate the run-time environment for that program. These statements include:

- //EXECUTE
- //PAR
- //DEFINE
- //SET
- //TELL

### EXECUTE STATEMENT

```
//EXECUTE
//EXEC
//EX      PGM=program-name
          [ { NAME } =step-name ]
          [ { NAM } =step-name ]
          [ { LIBRARY } =load-library-name ]
          [ { LIB } =load-library-name ]
          [ { TIME } =minutes ]
          [ { TIM } =minutes ]
          [ { DUMP } = { YES } ]
          [ { DMP } = { NO } ]
          [ { COND } ]
          [ { DEBUG } = { YES } ]
          [ { DEB } = { NO } ]
          [ { RESTART } = { YES } ]
          [ { RES } = { NO } ]
          [ { nnn } ]
```

The //EXECUTE statement is the first statement of every step. It identifies the program to be executed, names the job step, and specifies the library on which that program resides. Also indicated are the time that the step is to be allowed to run and the conditions under which a dump is to be made.

**PGM=program-name**

**(required)**

This keyword specifies the name of the program to be executed. The program name is a 1- to 8-character alphanumeric field, which is the same name as that by which the program is cataloged on the library. If RESTART=nnn option is specified, PGM=RESTART must be specified.

**{ NAME }  
{ NAM } =step-name**

**(optional)**

The system allows job steps to be named and identified by the NAME keyword. If the step is to be referenced by the GO keyword of a preceding //IF statement, this keyword-operand is required. (The //IF statement is described later in this section.) The operand is a 1- to 8-character alphanumeric field.

**{ LIBRARY }  
{ LIB } =load-library-name**

This keyword specifies the library on which the program to be executed resides. It is required whenever the program resides on a private load library rather than on the standard system load library (\$SYSLODLIB). The operand is a 1- to 17-character alphanumeric field.

If the library is centrally cataloged, no //DEFINE statement is required for the library. Otherwise the //DEFINE statement must be included with ID=\$LODLIB.

**{ TIME }  
{ TIM } =minutes**

**(optional)**

This keyword-operand gives the maximum time, in minutes, that the step is allowed to run. If the operand defined is executed, the step will be terminated abnormally and the job will be aborted.

A 4-digit numeric field is used for this operand. Maximum time that can be specified is 1440 minutes, which allows the step to run indefinitely. If 1439 minutes is specified, the step will be allowed to run for 23 hours 59 minutes before being aborted. The default is set by the installation at SYSGEN time.

**{ DUMP }  
{ DMP } = { YES  
          NO  
          COND }**

**(optional)**

The system allows the conditions for making a main storage dump to be specified. When DUMP=YES, a dump of the user partition is made at the end of the step regardless of the termination status (normal or abnormal). When DUMP=NO, the dump is not made regardless of the termination status. If the keyword is not specified or if DUMP=COND, a dump will be made upon abnormal termination of the step, but not upon normal termination.

$\left\{ \begin{array}{l} \text{DEBUG} \\ \text{DEB} \end{array} \right\} = \left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\} \quad \text{(optional)}$

This keyword\* allows the programmer to call the Debug routine for use in his program. This utility provides for snapshot dumps and optional step termination at specified breakpoints.

When DEBUG=YES, Control Language Services opens the Debug directive file created with a Control Language //DATA statement. Control Language Services will also inform the Relocating Program Loader that the Debug utility is to receive control over processing. When DEBUG=NO, the program is not run in Debug mode. The default is NO.

$\left\{ \begin{array}{l} \text{RESTART} \\ \text{RES} \end{array} \right\} = \left\{ \begin{array}{l} \text{YES} \\ \text{NO} \\ \text{nnn} \end{array} \right\} \quad \text{(optional)}$

This keyword indicates a step's eligibility for restart<sup>†</sup> after abnormal termination. RES=YES allows immediate restart if a step terminates abnormally; RES=NO does not.

The nnn operand is a 1- to 3-digit decimal number indicating location for deferred restart attempt. The nnn value is taken from the console run sheet or the SYSOUT listing of the previous run of this step. If a deferred restart is being attempted, PGM=RESTART must be specified in addition to the RESTART keyword specification in the //EXEC statement.

Whenever RESTART is specified in the //EXEC statement, a //DEFINE statement with ID=SYSCHK for the checkpoint file must also be included in the Control Language statements.

### Sample Execute Statement

The following is an example of an //EXECUTE statement. The required keyword, PGM, and some optional keyword-operands are used.

```
//EXEC    PGM=LOD618,DEBUG=YES,LIB=ULIB20,  
//        NAM=STEP32,TIM=80
```

In this example, the program LOD618 will be executed from the library named ULIB20. The step name is STEP32 and may be branched to by a preceding //IF statement. The program will be allowed to run 80 minutes before being aborted. The Debug routine will run with this step. Breakpoints given in the Debug data directive file that apply to this step will be activated. Since DUMP is not specified, a dump will be taken on abnormal termination, but not on normal termination of the step.

\*All programmers using the DEBUG keyword must be familiar with the Debug routine and the contents of the Debug directive file. This information is available in the Control Program and Data Management Services, Basic Reference manual.

<sup>†</sup>Restart information is discussed in the MRX/OS Control Program and Data Management Services, Basic Reference manual.

## PAR STATEMENT

Run time parameters are supplied to system programs and user programs through //PAR statements. The parameters listed on //PAR statements are read by Control Language Services and transferred to the SYSIN file. //PAR statement parameters which are included in the job stream for the execution of a system program (such as the COBOL Compiler) are read by the system program itself. Those parameters supplied by //PAR statements to be used by the problem program require an ACCEPT macro to be read from the SYSIN file. (Appendix D contains a brief description of the ACCEPT macro; further details can be found in the **Control Program and Data Management Services, Extended Reference** manual.)

The specified parameters in the //PAR statement may have the same form as other control language statements, using keyword-operands, or they may take any other form acceptable to the program using them. Regardless of the form of the parameters, the identifier, command, and sequence fields must conform to the format specifications of all other control language statements. *The //PAR statement may be continued, but each statement must begin with //PAR.*

### Sample //PAR Statements

The following examples show //PAR statements which might be included in the control language statements of a step:

```
//PAR OP=SMAP,KEY=124311,DECK=YES  
//PAR BRANCH=ABK,  
//PAR PAY:TYPE42/CONT  
//PAR SPECIAL-ORDINARY
```

In these examples, keyword and other operand forms are used. The program executed in the step uses the ACCEPT macro to obtain the //PAR statement from the SYSIN file and processes the operands internally.

### NOTE

A special restriction applies to //TELL and //PAR statements: No more than 17 characters may appear in consecutive columns without a delimiter (blank, comma, equal sign).



**DEFINE STATEMENT**

//DEFINE  
//DEF

{ IDENTIFIER } =file-identifier  
{ IDENT  
ID }  
{ FILENAME } =name  
{ FIL }

{ STATUS } = { S  
T  
W }  
{ STA } = { ( P [ I  
O  
U ] ) }

[ MSC=code ]

[ { DEVICE } = { name  
DEV } = { (name,quantity)  
address(es) } ]

[ { VOLUME } = { volume-id  
VOL } = { (volume-id,volume-id,..) } ]

[ CSD= { YES  
NO } ]

[ { ORGANIZATION } = { S  
ORG } = { R  
I } ]

[ { LABEL } = { S  
LAB } = { N  
U  
I  
B } ]

[ { RETENTION } =number  
RET ]

[ { BUFFER } = (number,size)  
BUF ]

[ { NUMBER } = { n  
NUM } = { (n,e) } ]

[ { SIZE } = (record-size[,key-size])  
SIZ ]

[ { BLOCK } = (records-per-block[,keys-per-block])  
BLK ]

[ { LOCATION } = { YES  
LOC } = { NO  
n } ]

[ { CATALOG } = { YES  
CAT } = { NO } ]

[ { VERIFY } = { YES  
VER } = { NO } ]

[ { SPREAD } =n  
SPR ]

[ { CONTIGUOUS } = { YES  
CON } = { NO  
(( { SX } { SP }  
{ MX } { MP } )) } ]

[ { IVOLUME } =volume identifier  
IVOL }  
IVO ]

[ { ILOCATION } = { YES  
ILOC } = { NO  
n } ]  
ILO ]

[ { EXPAND } =n  
EXP ]

The unshaded areas in the preceding representation of the DEFINE statement apply to all file and device use, and disc space allocation. The shaded areas apply to disc space allocation and disc file expansion. Exceptions are noted in the text.

The //DEFINE statement is the run-time interface between Data Management and Control Language Services. The program is independent of the name of the file to be processed as well as the volume or device on which the file resides. Control Language Services performs all device assignments for steps and allocates space for new disc files from //DEFINE statements.

The //DEFINE statement specifies the files to be used and the device and volume requirements of a step. This statement may indicate the type of file, specify the use of a permanent file, define the attributes of the data file, and establish a logical relationship with the internal file definition of the program. The //DEFINE statement is required for a specific device or volume requested, and for file allocation.

Before coding //DEFINE statements, the programmer must be familiar with the contents of the **MRX/OS Control Program and Data Management Services, Basic Reference** manual for logical I/O processing and the **MRX/OS Control Program and Data Management Services, Extended Reference** manual for physical and block I/O processing. For the proper use of //DEFINE statements for telecommunications, the user should refer to the **MRX/OS Telecommunications Reference** manual.

I { IDENTIFIER } =file-identifier (required)  
  { IDENT  
  ID

The operand of this keyword supplies a logical name to the data file. That is, IDENTIFIER specifies the name by which the program will open, close, and transfer data to or from a data file. This file identifier must be unique within a job step.

The operand is a 1- to 8-character alphanumeric (including \$ and dash) field. SYSIN and SYSOUT are illegal operands for the IDENTIFIER keyword. This keyword-operand is required on all //DEFINE statements.

{ FILENAME } =name (required for disc and tape)  
  { FIL

This operand supplies label information. If the file is being allocated, the FILENAME operand will become the catalog name for disc data files, or the label entry for tape files.

The operand is a 1- to 17-character alphanumeric field. The first character may be A-Z, 0-9, or \$. Imbedded dashes are allowed, but not imbedded blanks in the succeeding characters. This keyword-operand is required for all tape and disc files, but does not apply to unit record files.

For all physical I/O, the FILENAME operand must be PIO. For telecommunication lines, except PIO, FILENAME=TP is used. Since the PIO specification requires the exclusive use of the device/volume requested, this specification is not legal for shared devices.

FILENAME=DUMMY indicates that label information and allocation attributes will be specified within the program. FILENAME=NULL specifies an optional file.

$$\left\{ \begin{array}{l} \text{STATUS} \\ \text{STA} \end{array} \right\} = \left\{ \begin{array}{l} \text{S} \\ \text{T} \\ \text{W} \\ \text{P} \end{array} \right\} \text{ or } \left\{ \begin{array}{l} \text{STATUS} \\ \text{STA} \end{array} \right\} = \left( \text{P}, \left\{ \begin{array}{l} \text{I} \\ \text{U} \\ \text{O} \end{array} \right\} \right) \quad (\text{optional, disc only})$$

This keyword is used to specify the type of disc file used. The first operand specifies the type as scratch (S), temporary (T), work (W), or permanent (P). The second optional operand specifies usage for permanent files only. This usage may be specified as input (I), update (U), or output (O). The default for the first operand, which specifies type, is T. If the first operand is specified as P and the second operand is not specified, the default for it is O.

Detailed description of the file types and usage is given in **Control Program and Data Management Services, Basic Reference** manual. A brief description follows.

#### Scratch Files

Scratch files are allocated in a step and may be used only for the duration of that step. They are automatically purged at step termination. These files are always listed on the central catalog; therefore, CATALOG=NO is an illegal keyword-operand. Scratch files may never be shared.

#### Temporary Files

Temporary files are allocated in a step and may be used for the duration of the job. They are automatically purged at job termination. These files are always listed on the central catalog; therefore, CATALOG=NO is an illegal keyword-operand. Temporary files may never be shared.

#### Work Files

Work files are allocated permanently. They are not automatically purged, the Purge utility is used to perform this function. Work files may never be shared.

## Permanent Files

Permanent files are allocated permanently. They are not automatically purged; the Purge utility is used to perform this function. Permanent files may be shared, depending upon the usage operand specified for the file (Table 2-1):

- Input – when a usage of input is specified, the file may be opened only for input in that step. This file may be shared with programs requesting it for update or input in the other partition.
- Update – when a usage of update is specified, the file may be opened for input or update in that step. This file may be shared only with programs requesting it for input in the other partition.
- Output – when a usage of output is specified, the file may be opened for input, update, or output in that step. This file may never be shared with a program in another partition.

Table 2-1. Sharing of Permanent Files

//DEFINE File Usage	Allowable Shared Usage		
	INPUT	OUTPUT	UPDATE
INPUT	Yes	No	Yes
UPDATE	Yes	No	No
OUTPUT	No	No	No

**MSC=code**

**(optional)**

This keyword allows a modification security code to be specified for a permanent or work disc file. The code is used by Data Management to exclude unauthorized access to the file.

This keyword applies to both allocation and use of disc data files. The operand is a 4-character EBCDIC code. Embedded blanks and the characters ( ), = & may not be used. The default code is blanks.

$\left\{ \begin{array}{l} \text{DEVICE} \\ \text{DEV} \end{array} \right\} = \left\{ \begin{array}{l} \text{name} \\ \text{(name, quantity)} \\ \text{address(es)} \end{array} \right\} \quad \text{(optional)}$

This operand specifies the generic name and quantity or the device address(es) to be used for the file. The generic names of devices for MRX/OS are listed below. The default is DEV=DISC.

DISC  
TAPE8 (800 bpi)  
TAPE16 (1600 bpi)  
CRD or READER  
CRDPCH,RDRPUN,READPUNCH, or READERPUNCH  
PRT or PRINTER (600 or 1200 lpm)

SYSCRD is used to specify the system card reader whether it be a reader or reader-punch (see //DATA Statement FIL=SYSCRD).

In addition, the following generic names are used for telecommunication lines. Note that the last two characters are equipment types, defined in the **Telecommunications Reference** manual.

TP80  
TP84  
TP86  
TP88  
TP8A  
TPA4  
TPA6

The user may specify the device requirements by.

1. Generic name only; quantity of 1 is implied.
2. Generic name and quantity (quantity greater than 1 applies only to tape, disc, and TCOM; maximum quantity of 2 for tape).

Example: DEV=(TAPE8,2)

3. Address of a device as specified in the unit table.

Example: DEV=101

4. Addresses of the devices requested as specified in the unit tables.

Example: DEV=(302,303)

Specifying devices by device address instead of generic name allows two files to be applied to the same device.

| { VOLUME } = { volume-id  
| { VOL } = { (volume-id,volume-id,..) }

This specifies the volume identifier for file location. VOLUME is a required entry for disc files that are not listed on the central catalog. For tape files, VOLUME is required for the first or only volume of all files. In the case of multi-volume files, Data Management will write this volume identifier on each reel. In the case of multi-volume disc files, the user must specify the volume identifier. This keyword does not apply to unit record files. VOL=WORK may be used to specify a disc or tape volume with no volume identifier.

This operand is a 1- to 6-character field. It is an alphanumeric value given as the volume identifier.

CSD= { YES } (optional)  
| { NO }

This specifies whether or not data in the file is to be written in common stored data format. YES indicates that common stored data format is used; NO specifies that the data is not in common stored data format. The default is CSD=YES. If the common stored data format is used, four bytes are added to the record size for the control header.

{ ORGANIZATION } = { S } (optional)  
| { ORG } = { R }  
| { I }

This specifies the type of file organization as sequential (S), relative (R), or indexed (I). The system default is sequential file organization. (See the **Control Program and Data Management Service, Basic Reference** manual for additional discussion of file organizations.)

| { LABEL } = { S } (optional, tape only)  
| { LAB } = { N }  
| { } = { U }  
| { } = { I }  
| { } = { B }

| This specifies the type of tape label processing. The LABEL operand (valid for tape only) on the //DEFINE statement overrides any label specification for that file in the Buffer Description Table created by the Data Management DEFSS macro. The options are: standard labeled (S), nonstandard labeled (N), unlabeled (U), bypass label (B), and ignore tape label (I). (Refer to Section 3 of this manual and the **Control Program and Data Management Services, Basic Reference** manual for further details.)

**{ RETENTION } =number (optional, tape only)**  
**{ RET }**

This designates the number of days a tape file is to be retained before it is allowed to be purged. Data Management uses this information to calculate the expiration date to be stored on the file label. RETENTION is ignored for input files.

The operand for RETENTION is a 1- to 3-digit numeric field. The system default (zeros) is used when this keyword is not given. In that case, the expiration date on that file will be the same as the creation date.

**{ BUFFER } =(number,size) (optional, TCOM only)**  
**{ BUF }**

This keyword specifies partition space required for buffers. It has a two-part operand. The first is the number of buffers to be created for this line, and the second operand is the size of each buffer.

The number operand is a 1- to 3-digit numeric field with a value in the range of 1 to 999. The size operand is a 1- to 5-digit numeric field with a value in the range of 1 to 49999.

The buffers created with this keyword-operand are located in the partition space pool of the program partition in which the job runs and must not exceed the space available in the pool.

### Sample //DEFINE Statements

The following are examples of //DEFINE statements that provide for device assignment and for the use of already allocated disc files.

Sample 1: //DEF ID=SM220,DEV=PRT

In this example, a printer is assigned to the current job step. The file is called SM220 in the program.

Sample 2: //DEF ID=CLASS1,FIL=DA1143,VOL=136921,RET=3,  
// DEV=TAPE8,CSD=YES

In this example, a tape unit is assigned to the step with a tape volume label, 136921, and tape file label, DA1143. The tape file is assumed to have standard labels. The data is in a common stored data format on an 800 bpi unit. The volume is to be retained for 3 days. The program calls the file CLASS1.

Sample 3: //DEF FIL=MASTER6BQ3,ID=SOURCE2,STA=(P,O),  
// MSC=HR32

In this example, a permanent cataloged disc file is to be used. The file resides on shared resources but will not be shared because its usage is specified as output. The file is called SOURCE2 in the program. The catalog entry is MASTER6BQ3 with a modification security code of HR32. (The order of listed keyword-operands is free form.)

#### DISC ALLOCATION KEYWORDS

The following paragraphs describe additional //DEFINE statement keywords that are applicable to disc space allocation. VERIFY=YES may also be used for disc I/O to override the use of VER=NO at allocation. The SIZE and BLOCK keywords may also be used for the tape files, as noted in the discussion of those keywords below.

**{ NUMBER } = { n } (required, disc allocation only)**  
**{ NUM } = { (n,e) }**

This keyword-operand specifies the number of logical records for which space is to be allocated in a new file and optionally followed by the number of logical records to be stored in the catalog for dynamic expansion. If this operand is not given, or is zero, space allocation will not occur. If the expansion factor is omitted, zero is stored in the catalog. Dynamic expansion of sequential files occurs only at the logical I/O level.

The n and e operands are 1- to 6-digit numeric values. Use of the NUMBER keyword excludes the use of EXPAND in the same //DEFINE statement.

**{ SIZE } = (record-size[,key-size]) (optional, disc allocation and tape I/O)**  
**{ SIZ }**

This specifies the number of bytes per logical record for all file organizations and the number of bytes in the key for indexed files. For variable length records, the record size given is the maximum record size. If CSD=NO is not specified, the record size is incremented by four by the system.

Record size is a 1- to 4-digit numeric value in the range of 1 to 9999. Key size is a 1- to 3-digit numeric value in the range of 1 to 255.

This keyword may also be used for tape files. When specified in the //DEFINE statement and also within the program data transfer request, the specification within the program overrides the //DEFINE statement value.

The default for this keyword is 252 bytes, and applies to both tape and disc. (If CSD=YES the effect is a total record size of 256 bytes.)



$\left\{ \begin{array}{l} \text{BLOCK} \\ \text{BLK} \end{array} \right\} = (\text{records per block } [, \text{keys per block}])$  (optional, disc allocation, disc expansion, tape I/O)

This specifies the blocking factor to be used in all disc file organizations. Also the number of keys in each key index information block may be specified for indexed files. For variable length records, the data blocking factor designates the minimum number of records per block.

The operands for this keyword are the number of records in the disc data block and optionally the number of keys in the index block. Both operands of this keyword are 1- to 3-digit numeric fields. The value of each operand is in the range of 1 to 255. The default for disc is 1 record per block.

The keyword may also be used for tape files. In this case, only the first operand (number of records) is used. When specified both in the //DEFINE statement and also within the program data transfer request, the value given within the program overrides the //DEFINE statement value. The default for tape is 1 record per block.

A more detailed explanation of this keyword is given in **Control Program and Data Management Services, Basic Reference manual**.

$\left\{ \begin{array}{l} \text{LOCATION} \\ \text{LOC} \end{array} \right\} = \left\{ \begin{array}{l} \text{YES} \\ \text{NO} \\ n \end{array} \right\}$  (optional, disc allocation, disc expansion)

This defines the cylinder boundary for direct access storage space allocation. The numeric value is valid only if the VOLUME keyword is coded. The default is NO. The operand meanings are:

1. LOCATION=n specifies the cylinder number on which the file is to begin. The operand n is a 1- to 3-digit numeric value, with a range of 1 to 202.
2. LOCATION=YES specifies that the file is to begin on a cylinder boundary selected by Data Management.
3. LOCATION=NO allows the file to begin at any point on the pack.

$\left\{ \begin{array}{l} \text{CATALOG} \\ \text{CAT} \end{array} \right\} = \left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\}$  (optional, disc allocation)

This specifies whether or not the FILENAME and LOCATION of the file will be placed in the central catalog. CATALOG=YES specifies that the file entry will be placed on the system central catalog as well as the pack catalog. CATALOG=NO specifies that the file will not be entered on the central catalog. When CATALOG=NO is used, the VOLUME keyword is required.

CATALOG=NO applies only to work and permanent files at allocation time, and is illegal for scratch and temporary files. The system default is CATALOG=YES.

$\left\{ \begin{array}{l} \text{VERIFY} \\ \text{VER} \end{array} \right\} = \left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$  (optional, disc allocation, disc I/O)

This specifies burst-check. At allocation time, this option is recorded in the catalog entry. VERIFY=YES specifies that all writes to the file will be verified. VERIFY=NO specifies that no burst-check will occur. If this keyword is omitted, the system default (VERIFY=NO) will be used.

If VERIFY=NO is given at allocation time, the user may override this by specifying VERIFY=YES in the //DEFINE statement for the execution of a particular job step. This override will not, however, become a permanent characteristic of the file.

$\left\{ \begin{array}{l} \text{CONTIGUOUS} \\ \text{CON} \end{array} \right\} = \left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \\ \left( \left\{ \begin{array}{l} \text{SX} \\ \text{MX} \end{array} \right\}, \left\{ \begin{array}{l} \text{SP} \\ \text{MP} \end{array} \right\} \right) \end{array} \right\}$  (optional, disc allocation, disc expansion)

This keyword-operand applies to both disc space allocation and disc file expansion. It specifies whether the space request is for a segmented or unsegmented block of disc space, and if specified, whether the space may occupy more than one pack. Extent and pack are coded as SX (single extent), MX (multiple extent), SP (single pack), and MP (multiple pack). Any combination of extent and pack is allowed. CON=YES means (SX,SP); and CON=NO means (MX,MP).

The CONTIGUOUS option applies only to the current space request. File expansion always results in the addition of a segment (represented by an extent in the FDT for that file): it will not necessarily yield a totally contiguous file, even if CON=YES was coded for both allocation and expansion.

$\left\{ \begin{array}{l} \text{SPREAD} \\ \text{SPR} \end{array} \right\} = n$  (index files only)

This specifies the spread factor, or the frequency with which consecutive logical records of a file are written on a track. Use of the spread factor allows for faster sequential processing of records since disc access time is not lost in retrieving following records (Figure 2-1).

Operand n is a 1- to 2-digit numeric value in the range of 1 to 10 specifying a spread of 0-9. When this keyword is not used for an indexed file, logically sequential blocks of records are physically adjacent on a track.

Physical Block No.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Logical Block No.	1	5	9	13	2	6	10	14	3	7	11	15	4	8	12	16

Figure 2-1. Indexed File with Spread Factor of 4

$$\left. \begin{array}{l} \text{IVOLUME} \\ \text{IVOL} \\ \text{IVO} \end{array} \right\} = \text{volume-identifier} \quad (\text{index files only})$$

This specifies the volume on which the key index information file is to reside. If the IVOLUME keyword is not used for indexed files, the key index information file will be placed on the first volume specified in the VOLUME keyword-operand. The operand is a 1- to 6-character alphanumeric field.

$$\left. \begin{array}{l} \text{ILOCATION} \\ \text{ILOC} \\ \text{ILO} \end{array} \right\} = \left. \begin{array}{l} \text{NO} \\ \text{YES} \\ n \end{array} \right\} \quad (\text{index files only})$$

This specifies the cylinder boundary for the beginning of space allocation for the key index information file. It is ignored if IVOL is not used.

There are three operands for this keyword; the default is NO:

1. ILOCATION=n specifies the cylinder number on which the key index information file is to begin. The operand n is a 1- to 3-digit numeric value with a range of 1 to 202.
2. ILOCATION=YES specifies that the key index file is to begin on a cylinder boundary selected by Data Management Services.
3. ILOCATION=NO allows the file to begin at any position on the pack.

### Sample //DEFINE Statement for Disc Space Allocation

The following is an example of a //DEFINE statement that will result in the allocation of disc space:

```
//DEF ID=GATHER1,FIL=SLID321BG,STA=(P,O),  
//   NUM=1450,SIZ=100,BLK=10,  
//   ORG=S,CON=YES
```

In this example, space will be allocated for 1450 records of 100 bytes each, blocked 10 logical records to a physical block. The file will be a sequentially organized permanent cataloged file in common stored data format, and will reside on shared disc (as defined at Initial Program Load time).

The file is cataloged as SLID321BG with a blank modification security code and is known in the program as GATHER1. Since the STATUS keyword specifies output, a request for the same file by another partition (while the step in the first requesting partition is executing) will cause the step in the second requesting partition to suspend initiation until the first step has terminated. At that time, the requested file is available for use by the step in the second partition, and initiation is continued.

### DISC FILE EXPANSION KEYWORD

$\left\{ \begin{array}{l} \text{EXPAND} \\ \text{EXP} \end{array} \right\} = n$  (expansion of sequential files only)

This specifies the number of records by which an already allocated sequential file is to be expanded. This expansion takes place during step initiation. When the EXPAND keyword is used, the BLOCK keyword is required if its value is other than one (default for block keyword). The blocking factor specifies the same value as that given for that keyword when the file was allocated. The CONTIGUOUS and LOCATION keywords may also be used for file expansion. These keywords will apply only to the expanded area.

This operand is a 1- to 6-digit numeric value. Use of this keyword excludes the use of the NUMBER keyword in the same //DEFINE statement.

### Sample //DEFINE Statement for Expansion

The following is an example of a //DEFINE statement that will result in the expansion of an already allocated disc file:

```
//DEF FIL=PAY473ROLL,ID=PAY3,EXP=800  
//   BLK=40,CON=YES,STA=(P,O),MSC=14WR,  
//   VOL=BLY632
```

In this example, an already allocated disc file is expanded 800 records, blocked 40. The blocking factor must be the same as that of the existing file. The file is listed on the pack catalog of volume BLY632 as PAY473ROLL, with a modification security code of 14WR. The request is for contiguous space. The file being expanded is permanently allocated, with usage as output. The file will be called PAY3 in the program.

### Summary of //DEFINE Statement Keywords

Table 2-2 gives a summary of //DEFINE Statement keywords according to I/O level and device category.

#### ROUTE STATEMENT

```
//ROUTE
//RTE
```

{	IDENTIFIER	}	=file-identifier
{	IDENT	}	
{	ID	}	
[	{ FILENAME	= { name }	]
[	{ FIL	= { NULL }	]
[	{ DEVICE	= {	PRINTER
			PRT
			READERPUNCH
			READPUNCH
			RDRPUN
			CRDPCH
			device address
[	{ DEV	}	]
[	{ VOLUME	= { volume-identifier	]
[	{ VOL	= { (volume-id,volume-id,...) }	]
[	{ CONTIGUOUS	= { YES	]
		NO	]
		( { SX } , { SP } )	]
		( { MX } , { MP } )	]
		]	*
[	{ BLOCK	=n	]
	{ BLK	}	*
[	{ SIZE	=n	]
	{ SIZ	}	*
[	{ SPOOL	= { YES	]
		NO	]
		]	*
[	{ NUMBER	= { n	]
		(n,e)	]
		]	*
[	UCS=	{ name	]
		(name,FOLD,VER)	]
		(name,NOFOLD,NOVER)	]
		]	***
[	{ HOLD	= { YES	]
		NO	]
		]	**
[	{ SAVE	= { YES	]
		NO	]
		]	**
[	{ COPY	=nn	]
	{ COP	}	**
[	{ FORMS	=form-type	]
	{ FOR	}	***

\*Disc file allocation and disc file expansion parameters.  
\*\*Keyword passes information to the spooling function.  
\*\*\*Parameters for both spooler and unit record device.

Table 2-2. Summary of //DEFINE Statement Keywords\*

Device Category	Logical and Block I/O						Physical I/O
	Card Reader Punch or Printer	Tape	TCOM	Disc File I/O Only	Disc Space Allocation and I/O	Disc File Expansion and I/O	All Devices
Keyword							
ID	R	R	R	R	R	R	R
FIL		R <sub>1</sub>	R <sub>2</sub>	R	R	R	R <sub>3</sub>
STA		I <sub>15</sub>		O <sub>4</sub>	O <sub>4</sub>	O <sub>4</sub>	
MSC				O <sub>5</sub>	O	O <sub>5</sub>	
DEV	R	R	R	O <sub>6</sub>	O <sub>6</sub>	O <sub>6</sub>	R
VOL		R		O <sub>7</sub>	O <sub>8</sub>	O <sub>8</sub>	O <sub>9</sub>
CSD		O <sub>10</sub>			O		
ORG		I <sub>15</sub>		O <sub>12</sub>	O <sub>12</sub>	O <sub>12</sub>	
LAB		O		I <sub>15</sub>	I <sub>15</sub>	I <sub>15</sub>	
RET		O					
BUF			R <sub>11</sub>		I <sub>15</sub>		
NUM			I <sub>15</sub>	I	R	I	
SIZ		O <sub>13</sub>			R		
BLK		O <sub>13</sub>			R	R	
LOC					O		
CAT					O		
VER				O <sub>14</sub>	O	O <sub>14</sub>	
CON					O	O	
SPR					O		
IVOL					O		
ILOC					O		
EXP				I	I	R	

File and Device Use
}

Disc File Allocation
Disc File Expansion
}

Disc File Expansion
}

Key: blank = Ignored  
 R = Required  
 O = Optional  
 I = Illegal

Notes:

1. Not used for unlabeled tape
2. FIL=TP
3. FIL=PIO
4. Required if not temporary
5. Required if used at allocation and not used with LABDEF on the OPEN
6. Required if specific drive desired
7. Required if not cataloged.
8. Required if non-shared packs desired
9. Required for disc volume mounting
10. Input tapes only
11. Required on first //DEF for logical TCOM
12. Required if not sequential
13. Applies to tape when not specified in the OPEN request for the file
14. Use disc-write-check for this step; override VER=NO at allocation for this step only
15. Results are unpredictable.

The //ROUTE statement allows a file to be spooled for output or to provide information for the unit record device such as UCS and FORMS.

$\left. \begin{array}{l} \text{IDENTIFIER} \\ \text{IDENT} \\ \text{ID} \end{array} \right\} = \text{file-identifier} \quad \text{(required)}$

This operand establishes the logical relationship between the program's internal file definition and the external specifications of the //ROUTE statement.

The operand is a 1- to 8-character alphanumeric (including \$ and dash) field.

$\left. \begin{array}{l} \text{FILENAME} \\ \text{FIL} \end{array} \right\} = \left\{ \begin{array}{l} \text{name} \\ \text{NULL} \end{array} \right\} \quad \text{(optional)}$

This operand supplies label information. The FILENAME operand must be identical to the entry in the central catalog. If the file is being allocated, the FILENAME operand will become the name in the catalog. FIL=NULL may be used to specify an optional file. An input request for an optional file results in end of file; an output request is ignored. If FILENAME is not specified, Control Language Services creates a default filename which is a 3-byte day of year, 6-byte time of day, and a 1- to 8-byte identifier.

$\left. \begin{array}{l} \text{DEVICE} \\ \text{DEV} \end{array} \right\} = \left\{ \begin{array}{l} \text{PRINTER} \\ \text{PRT} \\ \text{READERPUNCH} \\ \text{READPUNCH} \\ \text{RDRPUN} \\ \text{CRDPCH} \\ \text{device address} \end{array} \right\} \quad \text{(optional)}$

This operand specifies the generic name or device address to be used for the output file for the spool function. Valid generic names are PRINTER, PRT, READERPUNCH, READPUNCH, CRDPCH. The default is DEV=PRINTER. If spooling was not selected at SYSGEN time, this is the device that will be assigned at Step Initiation.

$\left. \begin{array}{l} \text{VOLUME} \\ \text{VOL} \end{array} \right\} = \left\{ \begin{array}{l} \text{volume-identifier} \\ \text{(volume-id, volume-id, . . .)} \end{array} \right\} \quad \text{(optional)}$

This operand specifies the volume serial number(s) on which the data set is located or to be allocated. The operand is a 1- to 6-character alphanumeric field.

## ALLOCATION AND EXPANSION KEYWORDS

The following paragraphs describe the parameters necessary for the spool file allocation and/or expansion.

$$\left\{ \begin{array}{l} \text{CONTIGUOUS} \\ \text{CON} \end{array} \right\} = \left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\} \left( \left\{ \begin{array}{l} \text{SX} \\ \text{MX} \end{array} \right\}, \left\{ \begin{array}{l} \text{SP} \\ \text{MP} \end{array} \right\} \right) \quad (\text{optional})$$

This operand specifies whether allocation or expansion request is for contiguous space; and if specified, whether the space may occupy more than one pack. Extent and pack are coded as SX (single extent), MX (multiple extent), SP (single pack), and MP (multiple pack). Any combination of extent and pack is allowed. CON=YES means (SX, SP); and CON=NO means (MX, MP).

$$\left\{ \begin{array}{l} \text{SPOOL} \\ \text{SPL} \end{array} \right\} = \left\{ \begin{array}{l} \text{YES} \\ \text{NO} \end{array} \right\} \quad (\text{optional})$$

This keyword specifies whether output to be spooled or not. SPL=YES specifies spooling. SPL=NO specifies that the output goes directly to a peripheral device. Unless the default has been changed at SYSGEN time, the default is SPL=YES.

$$\left\{ \begin{array}{l} \text{BLOCK} \\ \text{BLK} \end{array} \right\} = n \quad (\text{optional})$$

This operand specifies the number of logical records per block. The value of each operand is in the range of 1 to 255. The default value is 1.

$$\left\{ \begin{array}{l} \text{SIZE} \\ \text{SIZ} \end{array} \right\} = n \quad (\text{optional})$$

This operand specifies the number of bytes per logical record. Record size is a 1- to 4-digit numeric value with a range of 1 to 9999. The default value is 134.

### NOTE

The user is encouraged to use even-byte record size to allow output writer to use multiple block reads.



$\left\{ \begin{array}{l} \text{NUMBER} \\ \text{NUM} \end{array} \right\} = \left\{ \begin{array}{l} n \\ (n,e) \end{array} \right\}$  (optional)

This keyword specifies the number of logical records for which space is to be allocated and optionally the number of logical records to be stored in the catalog for dynamic expansion.

The n and e operands are 1- to 6-digit numeric values. Unless specified otherwise at SYSGEN time, the default value is NUM=(1000,500).

## SPOOLING INFORMATION KEYWORDS

The following paragraphs describe additional //ROUTE statement keywords that are used to pass information to the spooling function or provide UCS or FORMS information to the unit record device.

$\text{UCS} = \left\{ \begin{array}{l} \text{name} \\ (\text{name}, \text{FOLD}, \text{VER}) \\ (\text{name}, \text{NOFOLD}, \text{NOVER}) \end{array} \right\}$  (optional)

This keyword specifies the 2- to 4-character name of the Universal Character Set residing on \$NUCLIB which is to be loaded prior to printing. The FOLD and VER options are available. Default values are a SYSGEN name, NOFOLD, and NOVER. If SPL=NO, the Step Initiator issues UCS mount requests and loads the UCS buffer; whereas if SPL=YES, the information is passed to the spooler. See the **MRX/OS Utilities Reference** manual for a detailed description of UCS.

$\left\{ \begin{array}{l} \text{HOLD} \\ \text{HLD} \end{array} \right\} = \left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$  (optional)

This keyword specifies whether the file is to be (HOLD=YES) or is not to be (HOLD=NO) placed on the spooler hold queue at step termination. Default is HOLD=NO.

$\left\{ \begin{array}{l} \text{SAVE} \\ \text{SAV} \end{array} \right\} = \left\{ \begin{array}{l} \text{YES} \\ \underline{\text{NO}} \end{array} \right\}$  (optional)

This keyword specifies whether the file is to be (SAVE=YES) or is not to be (SAVE=NO) purged after it has been printed or punched. Default is SAVE=NO and the file is purged after output completion.

$\left\{ \begin{array}{l} \text{COPY} \\ \text{COP} \end{array} \right\} = \text{nn}$  (optional)

This operand specifies a 1- or 2-digit decimal number which specifies the number of copies to be printed or punched. Default value is 1.

**{ FORMS }**  
**{ FOR }** =form-type **(optional)**

This operand specifies a 1- to 6-character alphanumeric form type which the operator will be requested to mount prior to printing or punching. Default is standard forms.

If SPL=NO, the Step Initiator issues forms mount requests; whereas if SPL=YES, the information is passed to the spooler.

### **SAMPLE //ROUTE STATEMENTS**

The following example assigns the READPUNCH to the job identified by ID=JOB18. The default filename is created. Since SPOOL was not specified the default is YES. Before punching the operator will be requested to use form 18.

Sample 1: //RTE ID=JOB18,DEV=READPUNCH,FORMS=18

In the following example the MASTERFILE will be allocated in noncontiguous space on volume BLY635 with blocking factor of 10. The number of logical records is 1500 with 500 reserved for dynamic expansion. The file is to be saved after printing.

Sample 2: //RTE ID=JOB20,FIL=MASTERFILE,VOL=BLY635,  
// BLK=10,NUM=(1500,500),SAVE=YES

### **SET STATEMENT**

//SET { DATE }  
{ DAT } =job-date  
  
and/or  
  
{ SWITCH }  
{ SWI } =switch-setting

The operating system permanently allocates space in the partition for a job control table. This table contains, among other things, a job date and a program option switch byte. The problem program has access to these fields which enable the programmer to obtain the date for processing and/or output, and to test the setting of the option switch.

At the beginning of the job, the job date field is set equal to the system date, and the bits of the program option switch byte are set to zero. The //SET statement enables the user to modify the contents of these two fields in the Job Control Table. Either the job date or the switches, or both, may be altered by one //SET statement. The effect of a //SET statement remains until //EOJ is encountered or until another //SET statement with the same keyword occurs within the job. Any number of //SET statements may be included within a job.

**DATE=job-date**

**(optional)**

This keyword-operand specifies the date to be used for date-dependencies in the job. The job date that is set need not be the current date. The operand field is a 6-digit numeric value in the form mmddy: mm is the month with a value in the range of 01 to 12, dd is the day of the month with a value in the range of 01 to 31, yy is the decade and year with a value in the range of 00 to 99.

Example: DATE=012472

## NOTE

A Julian date will be calculated by the system and will be returned when it is requested using a JDATE macro in a program. (Refer to Appendix D and the **Control Program and Data Management Services, Basic Reference** manual for further discussion of the JDATE macro.)

**SWITCH=switch-setting** **(optional)**

This allows the programmer to control the eight switch bits of the program option switch (POST) byte in the Job Control Table (JCT). The switches are set and changed by either the //SET control language statement or the POST macro and may be used for any purpose in the program. The switches are tested within the program via the RPOST macro. (See Appendix D of this manual and the **Control Program and Data Management Services, Extended Reference** manual for details on these macros.)

The operand is an 8-character field. Each character is specified as either a zero, a one, or an X. The value 1 sets the corresponding switch bit on in the POST byte of the JCT. The value 0 turns the switch bit off. The X leaves the switch bit as it was previously set within the job.

Example: SWI=0100X1XX

### Sample //SET Statements

The following are examples of //SET statements.

Example: //SET DATE=032172

In this example, the existing JDATE entry of the JCT will be changed to 032172. The SWITCH keyword-operand has been omitted and the POST byte of the JCT will not be affected by this //SET statement.

Example: //SET DATE=032173,  
// SWI=X0100XX1

In this example the JDATE entry of the JCT will be changed to 032173. The POST byte of the JCT is also changed. The first, sixth, and seventh bits retain the values set in them before this //SET statement is processed. The second, fourth, and fifth bits are turned off by being set to zeros. The third and eighth bits are turned on, by being set to ones.

## TELL STATEMENT

```
//TELL  
//TEL      OP=message  
           [ { PAUSE } = { YES }  
             { PAU  } = { NO } ]
```

This statement designates information to be typed on the operator's console during the initiation of a job step. The PAUSE=YES keyword-operand will cause Control Language Services to wait for an operator response before continuing initiation.

Continuation of this statement is not allowed. However, multiple //TEL statements may be used. The comment field is not allowed.

**OP=message** **(required)**

The operand of this keyword is the message to be printed on the operator's console. The entire EBCDIC code set is available for use in the operand. The contents of the operand will be typed on the operator's console as specified between the equal sign of the keyword and either the sequence field of the statement or the PAUSE=YES or PAU=YES keyword-operand.

Example: OP=EMPTY READER PUNCH HOPPER

```
{ PAUSE } = { YES }  
{ PAU  } = { NO } (optional)
```

PAUSE=YES indicates that the Control Language Services will wait for an operator response. The operator replies GO to continue processing or STOP to abort the job. When PAUSE=NO is specified or when this keyword is not coded, initiation will not be interrupted for operator response. The default is NO.

In order to write a multi-line message on the operator's console and then interrupt the initiation of the step for a response, PAUSE=YES is specified only on the last //TELL statement.

Special consideration must be given to the information that may be included in a //TELL statement. An equal sign within a message can cause Control Language Service errors. For example,

```
//TELL OP=PGM=XYZ,FIL=ABC
```

will cause an error. However, this may be circumvented by using the following format.

```
//TELL OP=* PGM=XYZ,FIL=ABC
```

## Sample //TELL Statements

The following are examples of //TELL statements.

```
Example: //TEL OP=PLEASE CALL BOB MILLER,EXT394
//TEL OP=IF THIS JOB ABORTS
```

In this example, a multi-line message is typed on the operator's console and initiation of the step will not be interrupted.

```
Example: //TEL OP=EMPTY CARD READER HOPPER,PAU=YES
```

In this example a single line message is typed on the operator's console. After the message is typed, initiation of the step will be suspended until the operator replies either GO to continue initiation or STOP to abort the job.

```
Example: //TEL OP=REMOVE PRINTED OUTPUT FROM PRINTER
//TEL OP=EMPTY READER AND PUNCH OUTPUT
//TEL PAU=YES,OP=CALL EXT 472 ON JOB COMPLETION
```

In this example, the three line message is typed on the operator's console before initiation of the step is interrupted. Initiation will resume when the operator replies to the PAU=YES keyword-operand with GO; termination will occur when the operator responds with STOP.

### NOTE

A special restriction applies to //TELL and //PAR statements: No more than 17 characters may appear in consecutive columns without a delimiter (blank, comma, equal sign).

## INTERSTEP LEVEL STATEMENTS

Between steps, the //IF statement may be used to affect the conditional execution of the following job steps.

### IF STATEMENT

```
//IF      { CODE }
          { COD  } =value
GO=      { step-name }
          { EOJ  }
```

The //IF statement provides for testing the condition code set by the program in the Job IF Code (JTIFC) byte contained in the job control table (JCT). As a result of the test in the //IF statement, the job will either continue with the next control statement or will skip (in a forward direction only) to a named //EXECUTE or to end of job. Only one code condition is tested in an //IF statement. However, any number of these statements may be used between steps.

**{ CODE } =value** **(required)**  
**{ COD }**

This 1-byte operand field specifies a value to be compared to the JTIFC byte of the JCT. The operand may contain any EBCDIC character. If the value given as the operand of the CODE keyword is equal to the value in the JTIFC byte, Control Language Services will initiate the step named as the operand of the GO keyword.

The value in the JTIFC byte is specified in the program, via the SETIF macro. (Refer to Appendix D of this manual and **Control Program and Data Management Services, Extended Reference** manual for a detailed description of this macro.)

**GO= { step-name }** **(required)**  
**EOJ**

The operand of the GO keyword specifies either the name of a following step or EOJ. If the condition code in the JCT (JTIFC byte) is equal to the operand of the CODE keyword, a forward jump will be made to the named step if GO=step-name, or to the end of job if GO=EOJ. The step-name specified must be identical to a NAME operand of a following //EXECUTE statement within the job. An unnamed step may not be referenced. If the GO keyword names a step which is not a forward reference, all remaining steps are skipped.

The GO operand is a 1- to 8-character alphanumeric field. This keyword-operand is required.

#### **Sample //IF Statement**

The following is an example of an //IF statement.

```
//IF CODE=M,GO=EOJ
```

In this example, if the value in the JTIFC byte of the JCT, set by the SETIF macro in the program, is equal to M, the remaining steps in the job will be skipped. If the value in JTIFC is not equal to M, the next control language statement in the job will be processed.

## PROCEDURE-ORIENTED STATEMENTS

To eliminate the duplication of writing control language statements for each step, these statements may be cataloged as a source library member. This member is known as a cataloged procedure. (Cataloged procedures are described in Section 4 of this document.) The statements that identify, communicate with, and cause the merging of a cataloged procedure are known as procedure-oriented statements. There are two of these:

`//CALL` (Requests the merging of a cataloged procedure from the procedure source library into the control language statements of a job step, and passes values to the procedure for variables identified in the `//DECLARE` statement.)

`//DECLARE` (Identifies variables used within the control language statements of a cataloged procedure.)

Cataloged procedures consist of a `//DECLARE` as the first statement followed by step or interstep control language statements (`//EXECUTE`, `//PAR`, `//DEFINE`, `//SET`, `//TELL` and `//IF`). No other control language statements except comments are allowed.

### DECLARE STATEMENT

```
//DECLARE
//DEC          [Required run-time variables]
               [Run-time default variables]
```

The `//DECLARE` statement is required as the first statement of every cataloged procedure having defaults. It may identify variables to be supplied to the procedure.

The Control Language statements within a cataloged procedure may include both constants and run-time variables. Constants are keyword-operands that remain unchanged for all executions of the procedure. Run-time variables are keyword-operands that may change with succeeding executions of the procedure. Run-time variables must be declared as keywords in the `//DECLARE` statement.

An ampersand (&) preceding an operand within a cataloged procedure identifies that operand as a run-time variable.

The keywords listed on the `//DECLARE` statement identify two classes of run-time variables. Keywords which are listed with no operands on the `//DECLARE` statement; these keywords must be specified with operands in the `//CALL` statement each time the cataloged procedure is called into execution. Keywords listed with operands on the `//DECLARE` statement supply default values for operands not specified at the time the procedure is called. These default values are set at the time the procedure is cataloged.



### Sample //DECLARE Statement

```
//DEC  A,B,C,D=DISC
//DEF  ID=&A,FIL=&B,
//      VOL=&C,DEV=&D
```

In the above example of a cataloged procedure: Keywords A, B, and C are listed on the //DECLARE statement with no operands. These operands, therefore, must be specified in the //CALL statement each time the procedure is called into execution. Keyword D is listed with the default operand DISC. If this operand is not overridden by a specification in the //CALL statement at the time this procedure is called into execution, &D in the //DEF statement will be replaced with the operand DISC.

### CALL STATEMENT

```
//CALL
//CAL  { PROC } =procedure-name
       [ { LIBRARY } =library-name ]
       [ LIB ]
       [Required run-time variables]
       [Default values to be overridden]
```

This statement names the cataloged procedure to be merged with the control language statements of a job. Run-time variables for that procedure are also given on the //CALL statement.

All required keywords (specified with the keyword only and no operand on the procedure //DECLARE statement) must be given on the //CALL statement. In addition, any //DECLARE default values to be overridden (specified with the keyword and operand on the //DECLARE statement) may also be given on the //CALL statement.

**{ PROC } =procedure-name (required)**  
**PRO**

This keyword-operand is required. It specifies the name of the cataloged procedure to be merged with the control language statements of a job. The operand is a 1- to 8-character alphanumeric field and must be identical to the member name of the procedure on the source library.

Example: PROC=RECON3

**{ LIBRARY }  
LIB** =library-name

This keyword is required whenever the procedure resides on a private library rather than on the system procedure library (\$SYSPROCLIB). It specifies the library on which the cataloged procedure resides. The operand is a 1- to 17-character alphanumeric field. First character may be A-Z, 0-9, or \$. No embedded blanks or special characters except dashes are allowed.

The PROC keyword and the LIBRARY keyword (if specified) must precede the required run-time variables and default values specified on the //CALL statement. They must also be in the order: PROC, LIB.

If the LIB keyword is not coded, the first run-time variable may not begin with LIB.

### Required Run-Time Variables

The required run-time variables supply the values for the run-time variables listed on the //DECLARE statement with keywords and no operands. If //CALL names a variable that is non-existent, it is ignored.

### Default Values to be Overridden

These entries provide values which override default values specified in the //DECLARE statement with keyword operand.

### Sample //CALL Statements

The following are examples of //CALL statements.

Sample 1: //CALL PROC=PAWN3B,LIB=PROCLIB1

In this example, PAWN3B is the cataloged procedure on library PROCLIB1 which will be merged with the Control Language statements of the step. The //DECLARE statement contains no required variables, and there are either no defaults given or all defaults are accepted for this execution of the procedure.

Sample 1: //CALL PROC=SEND241,SOURCE=FIL 42,  
// SPOT=467,LINES=20

In this example, the cataloged procedure is called SEND241 and resides on \$SYSPROCLIB, the system library for cataloged procedures. Defaults are being overridden, or entries are required for LINES, SOURCE, and SPOT (listed on the //DEC statement with no operands).

Sample 3: At run time those operands in the procedure identified by an ampersand (&), will be replaced by run-time variables. These run-time variables are listed on the //CALL statement, or as default values on the //DEC statement. Operands provided by //DECLARE and/or //CALL statements, when substituted into the Control Language statement of the cataloged procedure must not cause format errors in statement of test procedures. The format of the //CALL and //DECLARE statements is the same as that described under FORMAT previously in this section.

```
//JOB NAME=PAYROLL
//EX PGM=PR146G-A,TIME=75
//CALL PROC=PAYPROC1,LIB=PAYPROK,A=PROC1234,
// B=FIL147,C=AQ1439,D=(P,O)
//EOJ
```

The procedure PAYPROC1 contains the following code:

```
PAYPROC1
//DEC A,B,C,D,E=DISC
//DEF ID=&A,FIL=&B,
// VOL=&C,STA=&D,DEV=&E,
// NUM=1600,SIZ=128,BLK=1
```

At run time the operands are substituted into the //DEFINE statement, and it reads as follows:

```
//DEF ID=PROC1234,FIL=FIL147,
// VOL=AQ1439,STA=(P,O),DEV=DISC,
// NUM=1600,SIZ=128,BLK=1
```

## DATA LEVEL STATEMENTS

All data is entered into the system through the data level statement within the job. The data level statements are //DATA and /\* (data delimiter).

The user may enter data to the system in one of two ways: either by spooling the data to a sequential disc file to be read by one or more of the job steps, or by temporarily dedicating the system reader to his job through the use of SYSCRD. In either case, he must precede the //DATA statement with a //DEF statement to identify the file in each step that uses it. Spooling is generally preferred on two counts: it does not tie up the system reader which temporarily prevents the entry of other jobs, and it provides input data files which may be read by more than one job step. (However, in an environment using the reader-punch as primary input device, the SYSCRD file preloaded with blank cards is the user's only means of punching cards.)

## DATA STATEMENT

```
//DATA      { FILENAME } = { SYSCRD  
           { FIL       } = { data-file-name }  
           [ CLS= { YES } ]  
             { NO } ]  
           [ { BLOCK } =blocking factor for spooled data ]  
             { BLK } ]  
           [ { NUMBER } =number of records ]  
             { NUM } ]  
           [ { CONTIGUOUS } = { YES } ]  
             { CON } = { NO } ]
```

The //DATA statement is used to identify the data file which follows. Within a job, data files always precede the //EOJ statement. If the data is spooled more than one data file may be included in a job. //DATA statement is required for each data file in the job. A spooled data file is reusable; it may be read by more than one step of a job.

When spooling data, the data file(s) may appear any place in a job, following a complete statement, after the //JOB card and before the //EOJ card. The FILENAME keyword names the file to be built.

Only one SYSCRD //DATA statement specifying FIL=SYSCRD can be used in a job. This //DATA statement and its associated data file must immediately precede the //EOJ statement. Multiple data files are entered, in this case, by coding data delimiters following each set of data cards that the job processes as a file (Figure 2-2). Control of the card reader will be given to the job using the data file for the duration of the job. When the data delimiter is encountered, the card reader file is closed by the program. In order to read each additional set of data (following a data delimiter) the program must again issue an open request for the card reader, using the IDENT specified in the //DEFINE statement for that file, and must again close the file when the data delimiter is read.

The data delimiter, /\* (slash, asterisk, blank), must be used to end each data file including the last data file within the job; except when CLS=YES is specified, /\*CLS is the data delimiter. The //EOJ card immediately follows the last data delimiter in the job.

When the data cards are to be read directly from the system card reader by the program, the data file must immediately precede the //EOJ card; FILENAME=SYSCRD must also be given in the //DATA statement. In this situation, the card reader is controlled by the program, until the job has processed the data and has terminated. Only then is control of the card reader returned to the system so that Control Language Services can read and initiate additional jobs.

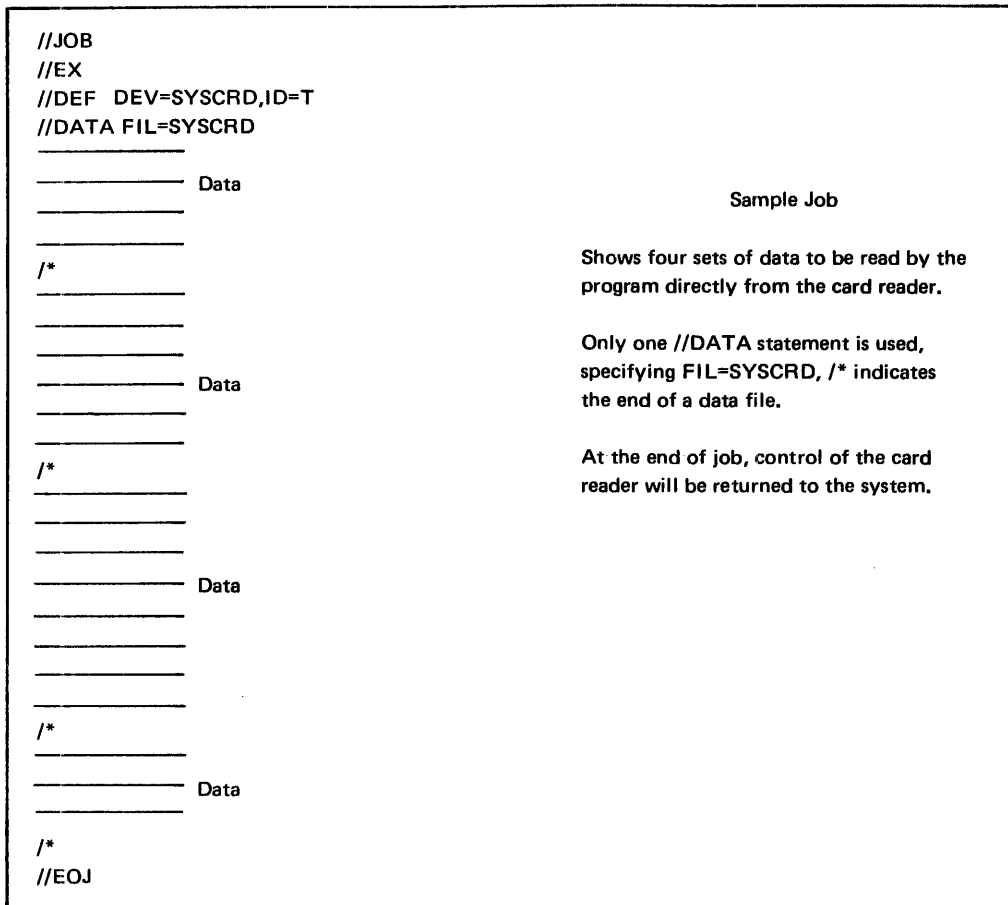


Figure 2-2. Sample Job with Card Reader Control by the Program

$$\left\{ \begin{array}{l} \text{FILENAME} \\ \text{FIL} \end{array} \right\} = \left\{ \begin{array}{l} \text{SYSCRD} \\ \text{data-file-name} \end{array} \right\} \quad \text{(required)}$$

This keyword specifies the name of the file to be built from the cards which follow. If data-file-name is used, this operand must be the same as that coded for the FILENAME operand of a //DEFINE statement within the job. If SYSCRD is used, the preceding //DEFINE statement uses DEV=SYSCRD rather than file-name.

When FILENAME=SYSCRD is used, control of the system card reader is passed to the job until all following data has been processed and the job has completed execution. The data-file-name operand is a 1- to 17-character alphanumeric field. First character may be A-Z, 0-9, or \$. No embedded blanks or special characters except dashes are allowed.

$\text{CLS} = \begin{Bmatrix} \text{YES} \\ \underline{\text{NO}} \end{Bmatrix}$  (optional)

The system allows control language statements to be placed in data files. Whenever this is done, as when creating procedure files, the CLS keyword is required. The CLS keyword-operand has no meaning when FIL=SYSCRD is used.

CLS=YES indicates that the control statement identifier (//) may be found in columns 1 and 2 of the records. The /\*CLS data delimiter is the only statement that will stop placement of cards in the data file. All cards between the //DATA and /\*CLS statements are spooled.

CLS=NO specifies that there are no control statements in the file. The first statement with the // or /\* identifier beginning in column 1 is assumed to be a control language statement and will terminate the file. The default is CLS=NO.

$\begin{Bmatrix} \text{BLOCK} \\ \text{BLK} \end{Bmatrix} = \text{blocking-factor for spooled data}$  (optional)

This operand specifies the blocking factor to be used in data spooling. It indicates the number of logical records to be placed in each physical block. The BLK keyword has no meaning when FIL=SYSCRD is used. The default is BLOCK=1.

$\begin{Bmatrix} \text{NUMBER} \\ \text{NUM} \end{Bmatrix} = n$  (optional)

This keyword is used to specify the number of logical records for which space is to be allocated for the spooled data file. If this operand is not given, the default value NUM=1000 is assumed. This parameter has no meaning if FIL=SYSCRD.

The NUMBER operand is a 1- to 5-digit numeric value. The maximum value of operand n is 32,767.

$\begin{Bmatrix} \text{CONTIGUOUS} \\ \text{CON} \end{Bmatrix} = \begin{Bmatrix} \text{YES} \\ \underline{\text{NO}} \end{Bmatrix}$  (optional)

This keyword specifies whether the space request is for a segmented or contiguous block of disc space for the spooled data file. CON=YES specifies that contiguous disc space is required. CON=NO indicates that segmented disc space is acceptable for this request. The default is NO. This parameter has no meaning if FIL=SYSCRD.

## Sample //DATA Statements

### Sample 1:

Example of data entry under program control of system card reader:

```
//JOB  NAME=SAMPLE
//EX   PGM=LIBUTIL
//DEF  ID=SEQIN,DEV=SYSCRD
.
.      (other //DEF and //PAR statements for program)
.
//DATA FIL=SYSCRD
.
.      (input data)
.
/*
//EOJ
```

### Sample 2:

Example of spooled data file, read by both steps of a two step job:

```
//JOB  NAME=GEMINI
//EX   PGM=FIRST
//DEF  ID=INPUT,FIL=PROC1630
.
.      (other //DEF and //PAR statements as required)
.
//EX   PGM=SECOND
//DEF  ID=PRIME,FIL=PROC1630
.
.      (other //DEF and //PAR statements as required)
.
//DATA FIL=PROC1630,CLS=YES,NUM=35,CON=YES
.
.      (input data)
.
/*
//EOJ
```

**DATA DELIMITER STATEMENT**

/\*

/\* (slash, asterisk, blank) is the data delimiter. This statement is required at the end of each data file.

There is no command for this statement and it has no keywords. The comment field may be used beginning in card column 4. Any character other than blank in card column 3 will cause the card to be treated as data and not as a data delimiter with the exception of /\*CLS.

If CLS=YES in the //DATA statement, all cards between the //DATA and /\*CLS cards are considered to be data.

**JOB STREAM CONVENTIONS**

A sample job stream is illustrated in Figure 2-3. Another sample may be found in Appendix G. Control language statements are listed as they might appear in a job. Within a job step the //EXECUTE statement must appear first. Other statements in the step may be in any order, but should be grouped by command. All statements in a job step are processed by Control Language Services prior to the program being loaded. Note the job stream conventions used.

Appendix G contains two sample jobs that include a variety of Control Language statements and appropriate keyword-operands to run these particular jobs.

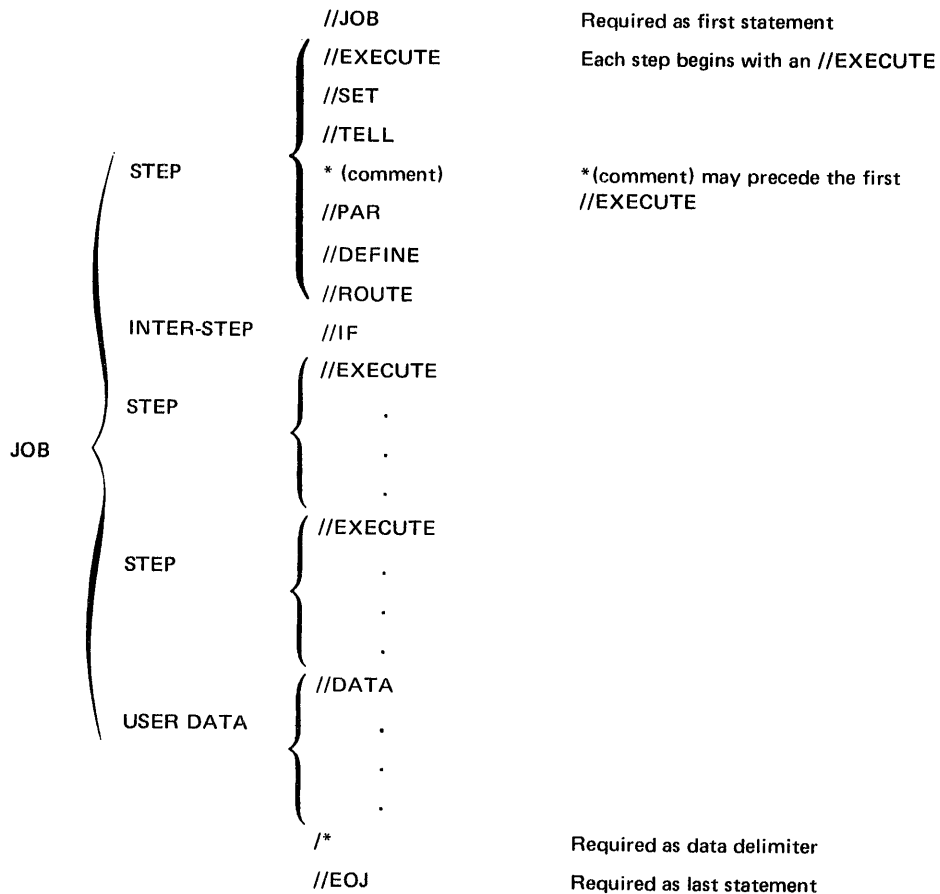


Figure 2-3. Sample Job Stream





## A. SUMMARY OF CONTROL LANGUAGE STATEMENTS

### JOB LEVEL STATEMENTS

Within a job, all control language statements are allowed. The following are required:	
//JOB	First statement of every job.
//EXECUTE //EXEC //EX	The first command following the //JOB statement, whether in line or as the first executable command (following //DECLARE) of a called procedure.
//EOJ	Last statement of every job.

### STEP LEVEL STATEMENTS

//EXECUTE //EXEC //EX	The first executable statement of every step; identifies the program to be executed.
The following statements may also be included:	
//PAR	Specifies run-time parameters to the program.
//DEFINE //DEF	Specifies devices, volumes, and files requested by the step.
//ROUTE //RTE	Allows output spooling or unit record device allocation.
//SET	Specifies job date and/or POST byte switch settings.
//TELL //TEL	Places messages on the operator's console.
//CALL	Merges Control Language statements from a cataloged procedure.
*	Comment statement.
Within a step, the following statements are not allowed:	
//JOB	First statement of every job.
//DATA //DAT	Defines the following data file. If data is spooled, it may appear anywhere between //JOB and //EOJ. If not spooled, it must immediately precede the //EOJ.
/*	Specifies the end of a card file.
//DECLARE //DEC	Designates the following control statements as a cataloged procedure, provides the specification statement for the procedure.
//IF	Provide for branching based on condition code test.
//EOJ	Last statement of every job.

### INTERSTEP LEVEL STATEMENTS

The only Control Language Statements occurring between steps are:	
//IF	Provide for forward branching based on condition code test.
*	Comment statement.

## CATALOGED PROCEDURES

Within a cataloged procedure, the following control language statement is required:	
//DECLARE //DEC	Specifies all keywords to be provided in calling the procedure; required as first statement of a cataloged procedure.
The following statements may also be included:	
//EXECUTE //EXEC //EX	The first executable statement of every step, identifies the program to be executed.
//PAR	Specifies run-time parameters to the program.
//DEFINE //DEF	Specifies devices, volumes, and files requested by the step.
//SET	Specifies job date and/or POST byte switch settings.
//TELL //TEL	Places messages on the operator's console.
//IF	Provide for forward branching based on condition code test.
*	Comment statement.
The following statements are not allowed:	
//JOB	First statement of every job.
//DATA //DAT	Identifies the following data file.
/*	Specifies the end of a card file.
//CALL //CAL	Calls a cataloged procedure.
//EOJ	Last statement of every job.
DATA LEVEL STATEMENTS	
A data file identified with the CLS=YES keyword on its //DATA statement may contain any control language statement, except /* (data delimiter) which terminates all data files. If CLS=NO on the //DATA statement, any control language statement except * (comment), will terminate the data file. CLS=YES has no meaning with FIL=SYSCRD.	

## B. TABLE OF CONTROL LANGUAGE STATEMENT KEYWORD CHARACTERISTICS

### //JOB Statement

Keyword	Operand Field		Default	Remarks
	Size	Content		
NAME= NAM=	1 to 8	Alphanumeric*	None – required entry	Specifies name of job as known to system. NAME keyword must appear on first card of statement. Jobs must have unique names.
USER= USE=	1 to 4	Alphanumeric*	None	Installation option. Required if option selected at SYSGEN time. Identifies the user to system.
TYPE= TYP=	1	0 1 2	0	Specifies partition in which is allowed to run: 0 indicates either partition 1 indicates partition 1 2 indicates partition 2 as created at SYSGEN time or as modified by console command.
PRIORITY= PRI=	1	1-9	SYSGEN default specified by installation	As specified at SYSGEN time. Specifies the order in which the jobs run in the partition. Highest priority is 9.
HOLD= HLD=	–	YES NO	No	HOLD=YES causes the job to remain in the job queue until the operator releases it for initiation.  HOLD=NO causes the job to be initiated according to its type and priority.

### //EOJ Statement

No keywords				Specifies end of job.
-------------	--	--	--	-----------------------

### Comment (\*) Statement

No keywords		* in column 1 followed by user comment		Enters comments into job stream.
-------------	--	--	--	----------------------------------

\*No embedded blanks or special characters except dash; alphabetic, numeric, and dollar sign in first character position.

**//EXECUTE Statement**

Keyword	Operand Field		Default	Remarks
	Size	Content		
PGM=	1 to 8	Alphanumeric*	None — required entry	Specifies the name by which the program to be executed is cataloged on the load library.
NAME= NAM=	1 to 8	Alphanumeric*	None	Required if step is to be named; whenever step is to be the argument of GO keyword in the //IF statement.  Specifies name by which step is known to system.
LIBRARY= LIB=	1 to 17	Alphanumeric*	Standard system library	Specifies name of library in which loadable program code resides.
TIME= TIM=	4	Numeric value in minutes	SYSGEN default specified by installation	Values in the range of 1 to 1440. Keyword specifies time in minutes the step is allowed to run. Value 1440 allows step to run indefinitely. Value 1439 allows step to run 23 hours 59 minutes before abort.
DUMP= DMP=	—	YES NO COND	Dump only for abnormal termination	DUMP=YES causes a main storage dump at end of step regardless of terminated status.  DUMP=NO results in no dump being taken regardless of terminated status.  DUMP=CON results in a dump only for abnormal termination.
DEBUG= DEB=	—	YES NO	No	Specifies if program is to run in Debug mode.
RESTART= RES=	Immediate — Deferred 1-3	YES NO Numeric	None	Specifies if the program can be immediately restarted. The nnn specifies location of deferred restart.

**//PAR Statement**

Application Dependent	<p>Example:</p> <pre>//PAR COMMAND=UPDATE, //PAR MTYPE=MAC, //PAR MEM=(,EMULATE), //PAR MEM=(,EMUCOM), //PAR MEM=(,EMUEQU), //PAR OLIB=(LIB3,SYM), //PAR NEWSEQ=(1000,100)</pre>	None	Contents of statement is application dependent.
-----------------------	--	------	---

\*No embedded blanks or special characters except dash; first character alphabetic, numeric, or dollar sign.

//DEFINE Statement

Keyword	Operand Field		Default	Remarks
	Size	Content		
IDENTIFIER= IDENT= ID=	1 to 8	Alphanumeric*	None — required entry	Establishes logical relationship between program file definition and physical characteristics of file. Keyword specifies name by which file is known in step.
FILENAME= FIL=	1 to 17	Alphanumeric*	None — required entry if labeled	Cataloged name or label entry for the file. PIO for physical I/O. TP for telecommunications. DUMMY for internal handling. NULL for optional file.
STATUS= STA=	—	S T W P (P,I) (P,U) (P,O)	T  O for P only	Applies only to disc. Type is Scratch, Temporary, Work, or Permanent. Input, Output, or Update usage is specified for permanent files only.
MSC=	4	EBCDIC	Blanks	Modification security code. Used by Data Management to exclude unauthorized use of file.
DEVICE= DEV=	—	Name (name , quantity) address (address,.... address)	(DISC,1)	Specifies generic name, and quantity (default quantity is 1), or unit address(es) for devices.
VOLUME= VOL=	1 to 6	Alphanumeric* valid (valid,.... valid)	Disc: central catalog Tape: none required entry	Identifies the disc pack or tape reel. Tape: required Disc: required for files not listed, or to be listed on central catalog
CSD=	—	YES NO	Yes	Specifies that common stored data format is (YES) or is not (NO) used on tape input, or disc files. Note: adds 4 to size if operand is YES.
ORGANIZATION= ORG=	1	I R S	S	I is indexed R is relative S is sequential
LABEL= LAB=	1	S N U B I		For tape files only: S is standard N is non-standard U is unlabeled B is bypass I is ignored N, B, and I can be used for tape input files only.

\*No embedded blanks or special characters except dash; first character alphabetic, numeric, or dollar sign.

**//DEFINE Statement (Continued)**

Keyword	Operand Field		Default	Remarks
	Size	Content		
RETENTION= RET=	1 to 3	Numeric	0	Specifies the number of days a tape file is to be retained.
BUFFER= BUF=	Number 1 to 3 Size: 1 to 5	Numeric	None — required for each logical	Logical TCOM only. Specifies buffers in partition. Number range of 1 to 999. Size range of 1 to 49999.

**//DEFINE Statement — Disc Space Allocation/Expansion**

NUMBER= NUM=	1 to 6	Numeric	None — required to allocate a new disc file.	USE ONLY TO ALLOCATE A NEW DISC FILE. Number of records for which space is to be allocated; optionally the number of records for dynamic expansion.
SIZE= SIZ=	—	Disc record length (record length, key length)  Tape record length	252	Number of bytes per logical record (maximum for variable length).  For indexed files, number of bytes in the key.
BLOCK= BLK=	1 to 3	Disc data block (data block, key block) Values in the range of 1 to 255	1	Specifies blocking factor. For variable length records, specifies minimum number of records.
LOCATION= LOC=	—	1- to 3-digit numeric	No	Specifies cylinder number on which file is to begin, or that file is to (YES) or is not required to (NO) begin on cylinder boundary.
CATALOG= CAT=	—	YES NO	Yes	Specifies that file is to be (YES) or is not to be (NO) cataloged (central catalog). No is illegal for temporary and scratch files.
VERIFY= VER=	—	YES NO	No	Specifies that write-disc-check is to be (YES) or is not to be (NO) used after each WRITE or PUT.
CONTIGUOUS= CON=	—	YES NO $\left( \begin{matrix} \{SX\} & \{SP\} \\ \{MX\} & \{MP\} \end{matrix} \right)$	No	Specifies that allocation or expansion space is (YES) or is not (NO) to be contiguous; and whether it may be on single pack or multiple packs.
SPREAD= SPR=	1 to 2	Numeric 1 to 10	1	Indexed files only. Frequency with which consecutive logical records occur on track.

**//DEFINE Statement – Disc Space Allocation/Expansion (Continued)**

Keyword	Operand Field		Default	Remarks
	Size	Content		
IVOLUME= IVOL= IVO	1 to 6	Alphanumeric*	First volume of file	Indexed files only. Volume on which key index resides.
ILOCATION= ILOC= ILO	—	1- to 3-digit numeric YES NO	No	Indexed files only. Specifies cylinder number on which key index is to begin (valid only if IVOL is used), or that key index is to (YES) or is not required to (NO) begin on cylinder boundary.
EXPAND= EXP=	1 to 6	Numeric	None	Number of records by which to expand an existing sequential file; libraries cannot be expanded. See Table 2-2.

**//ROUTE Statement**

IDENTIFIER= IDENT= ID=	1 to 8	Alphanumeric	None – required entry	Establishes logical relationship between program file definition and external characteristics of file.
FILENAME= FIL=	1 to 17	Alphanumeric or NULL	3-byte day of year, 6-byte time of day, 1- to 8-byte identifier	Catalog name of file. FIL=NULL specifies an optional file.
DEVICE= DEV=	—	PRINTER PRT READPUNCH READERPUNCH RDRPUN device address	PRINTER	Specifies the generic name or device address of the output file for the spool function.
VOLUME= VOL=	1 to 6	Alphanumeric	None	Specifies the volume serial number(s) for the data set.

**//ROUTE Statement – Disc Space Allocation/Expansion**

SPOOL= SPL=	—	YES NO	YES	Specifies the file to be spooled (YES) or to go directly to the device (NO).
NUMBER= NUM=	1 to 6	Numeric	(1000,500)	Number of records for which space is to be allocated; optionally the number of records for dynamic expansion.
SIZE= SIZ=	—	Disc record length	134	Number of bytes per logical record.

\*No embedded blanks or special characters except dash; first character alphabetic, numeric, or dollar sign.



**//ROUTE Statement – Disc Space Allocation/Expansion (Continued)**

Keyword	Operand Field		Default	Remarks
	Size	Content		
BLOCK= BLK=	1 to 3	Disc data block	1	Specifies blocking factor.
CONTIGUOUS= CON=	–	YES NO $\left( \left\{ \begin{array}{l} SX \\ MX \end{array} \right\}, \left\{ \begin{array}{l} SP \\ MP \end{array} \right\} \right)$	NO	Specifies that allocation or expansion space is (YES) or is not (NO) to be contiguous; and whether it may be on single pack or multiple pack.

**//ROUTE Statement – Spooling Information**

UCS=	–	name (name,FOLD, VER) (name,NOFOLD, NOVER)	(SYSGEN name, NOFOLD, NOVER)	Specifies Universal Character Set. See <b>MRX/OS Utilities Reference</b> manual for FOLD and VER options.
HOLD= HLD=	–	YES NO	NO	Specifies whether file is or is not to be placed on the spooler hold queue.
SAVE= SAV=	–	YES NO	NO	Specifies whether the file is to be or is not to be purged after it has been printed or punched.
COPY= COP=	1 or 2	Decimal number	1	Specifies the number of copies to be printed or punched.
FORMS= FOR=	1 to 6	Alphanumeric	Standard forms	Specifies the form type for the operator to mount prior to printing or punching.

**//SET Statement**

DATE= DAT=	6	mmddyy	System date (at beginning of job)	Specifies month, day and year. Set for duration of job or until another //SET DATE.
SWITCH= SWI=	8	0 1 X	Zeros (at beginning of job)	Zeros change corresponding bit positions to zeros. Ones change corresponding bit positions to ones. X's leave corresponding bit positions unchanged. Set for the duration of job or until another //SET SWITCH. Switches are set to zeros at beginning of job.

**//TELL Statement**

OP=	–	message	None – required entry.	Message to console operator.
PAUSE= PAU=	–	YES NO	No	Specifies initiation of step is (YES) or is not (NO) to be suspended until response is received from the console operator.

**//IF Statement**

Keyword	Operand Field		Default	Remarks
	Size	Content		
CODE= COD=	1	Any EBCDIC character	None – required entry	Value to be tested for, set in previous program by SETIF service request.
GO=	1 to 8	Alphanumeric*	None – required entry	Name of step to be skipped to if condition CODE value is met. EOJ if skipping all remaining steps of job.

**//DECLARE Statement**

Procedure dependent	– –	None	Refer to DEC and CALL statements in Sections 2 and 4.
	Example: <pre>//CALL PROC=XAMPLE, LIB=PROC1, PNAM=XYZ, MIN=5 // // Procedure "XAMPLE" might be: //DECLARE PNAM,MIN=10 //EX PGM=&amp;PNAM, // TIME=&amp;MIN</pre>		

**//CALL Statement**

PROC= PRO=	1 to 8	Alphanumeric*	None – required entry	Name of the procedure to be included with the control language statements of this job. PROC must be the first entry.
LIBRARY= LIB=	1 to 17	Alphanumeric*	&SYSPROCLIB	Specifies the private library on which the cataloged procedure resides.
	Example: <pre>//CALL PROC=EMUGEN,LIB=EMUPROCLIB, MACVOL=\$SYSPK, // OBJVOL=EMUVOL,OBJLIB=EMUOBJLIB, LODVOL=\$SYSPK, // LODLIB=\$SYSLODLIB,SYSVOL=\$SYSPK, NUM=2000</pre>			

\*No embedded blanks or special characters except dash; first character alphabetic, numeric, or dollar sign.

**//DATA Statement**

Keyword	Operand Field		Default	Remarks
	Size	Content		
FILENAME= FIL=	1 to 17	Alphanumeric*	None – required entry	Names the file to be built from records which follow. Specifies filename used on input //DEFINE statement. FIL=SYSCRD if data read by program directly from card reader.
CLS=	–	YES NO	No	Keyword specifies there are (YES) or are not (NO) control language statements in following data file. Has no meaning when FIL=SYSCRD is used.
BLOCK= BLK=	1 to 2	Value in the range 1 to 12	1	Specifies blocking factor used in data spooling. Ignored when FIL=SYSCRD.
NUMBER= NUM=	1 to 5	Numeric value	1000	Specifies number of logical records for which space will be allocated. Maximum is 32,767. Ignored when FIL=SYSCRD.
CONTIGUOUS= CON=	–	YES NO	No	Keyword specifies that there is (YES) or is not (NO) the requirement of contiguous space. Ignored when FIL=SYSCRD.

\*No embedded blanks or special characters except dash; first character alphabetic, numeric, or dollar sign.

**Statement: /\* (data delimiter)**

No Keywords			
-------------	--	--	--

## C. TABLE OF REQUIRED AND OPTIONAL KEYWORDS BY CONTROL LANGUAGE STATEMENT

### JOB LEVEL STATEMENTS

Statement	Keyword	Required or Optional	Notes
//JOB	NAME= NAM=	Required	
	USER= USE=	Required/ Optional	Installation option. Required if installation requires at SYSGEN time. Optional if installation does not require it.
	TYPE= TYP=	Optional	
	PRIORITY= PRI=	Optional	
	HOLD= HLD=	Optional	
//EOJ	None	Required	End of job
* (comment)	None	Optional	

### STEP LEVEL STATEMENTS

//EXECUTE //EXEC //EX	PGM=	Required	
	NAME= NAM=	Optional	No default
	LIBRARY= LIB=	Optional	No default
	TIME= TIM=	Optional	
	DUMP= DMP=	Optional	
	DEBUG= DEB=	Optional	
	RESTART RES=	Optional	PGM=RESTART must be specified in //EXEC statement; may specify either immediate or deferred restart.
//PAR	Application dependent	Optional	No default; no continuation. Multiple statements acceptable.
//DEFINE //DEF for file and device use	IDENTIFIER= ID=	Required	
	FILENAME= FIL=	Required	Required for disc and tape; DUMMY, NULL, PIO can all refer to unit record.

Statement	Keyword	Required or Optional	Notes
	STATUS= STA=	Optional	Disc files only; usage for permanent files only.
	MSC=	Optional	
	DEVICE= DEV=	Optional	
	VOLUME= VOL=	Optional	Tape and disc files
	CSD=	Optional	Tape and disc files
	ORGANIZATION= ORG=	Optional	
	LABEL= LAB=	Optional	Tape files only
	RETENTION= RET=	Optional	Tape files only
	BUFFER= BUF=	Optional	Logical TCOM only. Requires 1 per line.
//DEFINE //DEF (for disc space allocation and expansion)	NUMBER= NUM=	Required to allocate a new disc file.	Use indicates space to be allocated. EXP may not be used
	SIZE= SIZ=	Required for disc if NUM is used. Optional for tape.	Keysize for indexed disc files only.
	BLOCK= BLK=	Optional	Applies to tape and disc files. Key block for indexed disc files only.
	LOCATION= LOC=	Optional	
	ORGANIZATION= ORG=	Optional	
	CATALOG= CAT=	Optional	
	VERIFY= VER=	Optional	
	CONTIGUOUS= CON=	Optional	Allocation and expansion
	SPREAD= SPR=	Optional	Indexed files only
	IVOLUME= IVOL=	Optional	Indexed files only
	ILOCATION= ILOC=	Optional	Indexed files only

Statement	Keyword	Required or Optional	Notes
//DEFINE (disc file expansion)	EXPAND= EXP=	Required to expand an existing disc file.	Use requires BLOCK equal to value specified at file allocation. Excludes NUMBER keyword.
//ROUTE RTE=	IDENTIFIER= IDENT= ID=	Required	
	FILENAME= FIL=	Optional	NULL specifies optional file.
	DEVICE= DEV=	Optional	
	VOLUME= VOL=	Optional	
	CONTIGUOUS= CON=	Optional	
	BLOCK= BLK=	Optional	
	SIZE= SIZ=	Optional	
	SPOOL= SPL=	Optional	
	NUMBER= NUM=	Optional	
	UCS=	Optional	
	HOLD= HLD=	Optional	
	SAVE= SAV=	Optional	
	COPY= COP=	Optional	
FORMS= FOR=	Optional		
//SET	DATE= DAT=	Optional	
	SWITCH= SWI=	Optional	
//TELL= //TEL=	OP=	Required	No continuation; multiple statements acceptable.
	PAUSE= PAU=	Optional	

PROCEDURE LEVEL STATEMENTS

Statement	Keyword	Required or Optional	Notes
//DECLARE //DEC	Procedure dependent	—	Statement may contain no keywords. Operands are required only to specify defaults.
//CALL //CAL	PROC= PRO=	Required	
	LIBRARY= LIB=	Optional	
	Procedure dependent	—	Required for all keywords listed on procedure //DECLARE without operands. Otherwise, used with operands only to override defaults.

INTERSTEP LEVEL STATEMENTS

//IF	CODE= COD=	Required	
	GO=	Required	

DATA LEVEL STATEMENTS

//DATA //DAT	FILENAME= FIL=	Required	
	CLS=	Optional	No meaning when FIL=SYSCRD used.
	BLOCK= BLK=	Optional	No meaning when FIL=SYSCRD used.
	NUMBER= NUM=	Optional	Default = 1000. No meaning when FIL=SYSCRD used.
	CONTIGUOUS= CON=	Optional	No meaning when FIL=SYSCRD used.
/* (Delimiter)	None		

## D. SYSTEM CONTROL INTERFACE

Control functions are provided for programs through Control Program service request macros. These macros are used in problem programs to obtain information from system and job tables, to access data that is set at run time with control language statements, to make the contents of the //PAR statement available to the executing program, and to write on the job's system output (SYSOUT) file.

The following paragraphs describe the Control Program service request macros referenced in this document. All programmers using these service requests must be familiar with the format and further explanation of the macros located in the **Control Program and Data Management Services, Basic and Extended Reference** manuals.

### HALT, EHALT, AND ABEND MACROS

The problem program signals the system that it has completed its processing, and causes the Step Terminator to be loaded by executing a HALT or EHALT service request macro. The HALT macro is used to perform normal termination of a user job step; the EHALT macro requests termination of a step and causes the remaining steps of a job to be skipped. The ABEND macro requests abnormal termination of a job and passes a completion code to Control Language Services for display.

When an error occurs (such as an I/O error, memory parity error, or attempt to write beyond the end of a program area), the program terminates abnormally. The Step Terminator is then loaded by the Control Program and receives control of the partition. A memory dump may occur, depending upon the dump keyword of the //EXECUTE statement.

### SDATE AND JDATE MACROS

The //SET control language statement may be used to specify the job date. This date is then used for date dependencies within the job. If the job date is not given with a //SET statement, the system date is used for these job dependencies.

The system date and the job date may be sent to the user program through the use of SDATE (system date) and JDATE (job date) macros respectively. The user may specify either the calendar date format or the Julian date format to be sent to his program. (Refer to **Control Program and Data Management, Basic Reference** manual for detailed description of these macros and formats.) If the job date has not been defined by a //SET statement it will be set to the system date by the operating system.



## **POST AND RPOST MACROS**

The //SET control language statement may be used to specify the contents of the switch communication byte (eight switches available to the problem program for communication between steps). The POST macro can be used to change the contents of the switch communication byte from within the problem program. The RPOST macro is provided to test the switch communication byte as set by previous POST macros or //SET control language statements.

## **SETIF MACRO**

The //IF control language statement tests the code condition of the JTIFC byte in the job control table (JCT) and conditionally executes job steps based upon the results. The SETIF macro is used to set this byte in the JCT to a code which may be tested by the //IF statement.

## **ACCEPT MACRO**

The //PAR control language statement enters run-time parameters for a problem program into the job's system input (SYSIN) file. The ACCEPT macro provides the problem program with the ability to read the steps //PAR statements into the program from the SYSIN file.

Each execution of the ACCEPT macro loads a single input line into the specified program buffer. The line is transferred into the buffer in EBCDIC. The macro updates a pointer in the requester's JCT so that subsequent ACCEPT macro requests will acquire the next consecutive parameter line. When accepting one line past the last parameter line, control is passed to the location specified in the ACCEPT macro (or the problem program may specify the relative parameter line number for that step).

## **DISPLAY MACRO**

The DISPLAY macro allows the problem program step to write messages directly onto the job's SYSOUT file. The message may be any length up to 132 EBCDIC characters. By using this macro, the problem program can indicate progress and internal program conditions on the SYSOUT file, which is printed at the end of the job.

## **MEMLIM MACRO**

The MEMLIM macro returns the address of the last addressable 256-byte memory page (256 bytes) of the partition making the request. The value returned points to the beginning of the last memory page of the problem program area.

The size of the problem program area, and therefore the address returned from a MEMLIM request, can vary depending upon the size and starting address of the partition and upon the specifications given to the Linkage Editor when the relocatable program load module is constructed.

Mnemonic	Bytes	Bits	Description
JTDBI	14,15	0-7	Debug Instruction. First two bytes of the instruction overlaid by the Debug service request. Set by Debug program. Used by System Control Program.
JTLPK	16-27	0-7	Loader Packet. Required for calling Program Loader. Modified by Control Language Services, System Control Program, and Program Loader. Used by System Control Program and Program Loader; and divided as follows:
FC	16	0-7	Function Code 10 <sub>16</sub>
LENGTH	17	0-7	Length of this Packet 7 words
C	18	0	Request Complete Indicator 0 Request has not completed 1 Request has completed
A	18	1	Completion Status Indicator 0 Normal Completion 1 Abnormal Completion
END	18	2	END bit 0 File End Condition has not occurred 1 File End Condition has occurred
Reserved	18	0-7	Reserved for system use
RETURN	19	0-7	Return Code 1 I/O Error 2 Cannot locate module via Entry Point List 3 Cannot locate module via library catalog 4 Module exceeds Partition limits 5 Invalid Load Address
Reserved	20 21	0-7 0-4	Reserved for system use.
SFC	21	5-7	Subfunction Code 0 Load by Entry Point 1 Load by Entry Point at Load Address 2 Load by Module Name 3 Load by Module Name at Load Address 4 Fetch by Entry Point 5 Illegal 6 Fetch by Module Name 7 JMSI Fetch Request (Control Language Services only)
Reserved	22,23	0-7	Reserved for system use.
NAMEADR	24,25	0-7	Address pointing to 8 byte Module or Entry Point Name.
Reserved	26,27	0-7	Reserved for system use.
LOADADR	28,29	0-7	Relative load address (if SFC = 1 or 3 only).
JTCOM	30,31	0-7	Completion Code. Conveys completion status of problem program. Set by System Control Program. Used by Control Language Services, Dump, and System Control Program.

Mnemonic	Bytes	Bits	Description
JTPOST	32	0-7	Program Option Switch Table. Provides inter-program switch testing and setting capabilities. Set by Problem Program and Control Language Services. Used by Problem Program.
JTIFC	33	0-7	Job IF Code. Any EBCDIC value that is set by a Problem Program via an SR. Tested by the Control Language //IF statement.
JTCLS	34	0	Internal switch for Job Initiator abort condition.
		1	Immediate restart requested.
		3-7	Job Class. Used to indicate in which partition the user wants a job to run. Set and used by Control Language Services.
JTPRI	35	0-7	Job Priority. Conveys priority of job, 1-9, as specified on the //JOB statement. Set and used by Control Language Services.
JTNAM	36-43	0-7	Eight character job name (left justified space filled if less than eight characters) specified on the //JOB statement. Set by Control Language Services. Used by Data Management and Control Language Services.
JTSTP	44-51	0-7	Eight character step name (left justified space filled if less than eight characters) specified on the //EXECUTE statement. Set and used by Control Language Services.
JTACC	52-55	0-7	Account Number. Specified as user account number on //JOB statement. Used for accounting purposes. Set and used by Control Language Services.
JTSTIM	56-59	0-7	Step Start Time. Packed decimal form HHMMSS. Set and used by Control Language Services.
JTETIM	60-63	0-7	Step Elapsed Time. Packed decimal form HHMMSS. Set and used by Control Language Services.
JTLIM	64,65	0-7	Step Time Limit. Maximum number of minutes the user wishes to allow his program to run. Set by Control Language Services. Modified and used by System Control Program.
JTJCS	66,67	0-7	Next Job Control Statement. Block number of next record in SYSIN. Set and used by Control Language Services.
JTDEF	68,69	0-7	First DEFINE. Block number of first DEF record in the SYSIN file for a step. Set by Control Language Services. Used by Data Management and Control Language Services.
JTPAR	70,71	0-7	Next Parameter. Block number of next PAR record in the SYSIN file for a step. Set by Control Language Services. Used by System Control Program.

## F. INDEX—BLOCK SIZE FOR INDEXED FILES

### CALCULATING BY TABLE

The minimum index block size and optimum block size may be calculated by Tables F-1 through F-4. Refer to the **MRX/OS Control Program and Data Management Services, Extended Reference** manual for the layouts of the index portion of indexed files.

### MINIMUM INDEX BLOCK

There is a *minimum index block size* for every indexed file depending on key size and file size. The user may utilize any index block size larger than the minimum, if he has memory space for a larger index block. The larger the index block the better retrieval becomes on random processing. If the user goes below the minimum index block size there is the possibility of not being able to create the file size as planned.

### OPTIMUM INDEX BLOCK

When planning the creation of indexed files, the user must decide whether he wants to process the directory-directory, which resides on mass storage, in a main memory buffer. This option speeds up random processing, but requires extra space for the buffer. If the mode of processing is with a main-memory buffer there is a well-defined *optimum index block size* which minimizes memory space for the index buffer and directory-directory buffer.

Once the user has determined his mode of processing, Table F-1 is used to determine minimum-keys/block and Table F-2 is used to determine optimum keys/block. Note that in using Table F-1 and Table F-2, the larger of the two values in the file size is the determining factor. Also note that these tables were computed for consistency for maximum key size and one million records as the upper limit. There will be some index block sizes generated that exceed one track in number of bytes. This exceeds the system limit for block sizes (limit is 7294 bytes). The user will have to choose a smaller key size or smaller file size.

### PROGRAMMING CONSIDERATIONS

The keys/block is entered in the Control Language //DEFINE statement along with key size. The corresponding minimum or optimum index block size can be calculated from Table F-3. The resulting index block size is then entered in the source program.

If the user has determined to calculate the optimum keys/block and optimum index block size, Table F-4 is used to calculate the number of bytes for the main-memory buffer for the directory-directory entries.

The user must be careful not to exceed the file maximum at creation time when using the optimum block size — when he utilizes the main-memory buffer to hold the directory-directory entries for random processing, the buffer would not be able to hold all the entries, thus writing over the user program. Thus, when choosing an index block size other than the optimum and the main-memory buffer is used to process the directory-directory entries, the buffer size should be the size of the index block, as the system checks for overflow at creation time.

## **EXAMPLES**

The following examples illustrate how to calculate the minimum index block size, the larger than minimum index block size, and the optimum index block size.

### **MINIMUM INDEX BLOCK SIZE**

The minimum index block size can be calculated with the following steps.

1. In Table F-1 locate the number of records in the file and the key size. For example, if the number of records is 20,000 and the key size is 10, the minimum keys per block is 24.
2. The keys/block is entered in the Control Language //DEFINE statement along with the key size.
3. The corresponding minimum index block size is calculated from Table F-3 using the minimum block size formula. For this example with minimum keys/block of 24 and key size of 10; the minimum index block size is 384 bytes.
4. The minimum index block size is then entered into the source program.

### **LARGER THAN MINIMUM INDEX BLOCK SIZE**

Similar to the minimum index block size, an index block that is larger than the minimum may be calculated with the same steps. The difference is found in estimating the keys/block, it must be greater than or equal to the minimum keys/block selected.

### **OPTIMUM INDEX BLOCK SIZE**

The optimum index block size can be calculated with the following steps.

1. In Table F-2 locate the number of records in the file and the key size. For example, if the number of records is 20,000 and the key size is 10, the optimum keys per block is 30.

2. The keys/block is entered in the Control Language //DEFINE statement along with the key size.
3. The corresponding optimum index block size is calculated from Table F-3 using the optimum block size formula. For this example with optimum keys/block of 30 and key size of 30, the optimum index block size is 475 bytes.
4. The optimum index block size is then entered into the source program.
5. For optimum keys/block and optimum index block size, Table F-4 is used to calculate the main-storage buffer for the directory-to-the-directory entries. For this example, the number of bytes required for the buffer for the directory-to-the-directory is 250.

Table F-1. Minimum Keys/Block

Records in File	Key Size in Bytes															
	2	3	4	5	6	7	8	9	10	11 to 15	16 to 20	21 to 25	26 to 35	36 to 50	51 to 100	
0 - 5,000	13	14	14	14	15	15	15	15	15	16	16	16	16	17	17	
5,000 - 10,000	16	17	18	18	18	19	19	19	19	20	20	20	21	21	21	
10,000 - 15,000	19	20	20	21	21	21	22	22	22	23	23	23	23	24	24	
15,000 - 20,000	20	21	22	23	23	23	24	24	24	25	25	26	26	26	26	
20,000 - 25,000	22	23	24	24	25	25	25	26	26	27	27	27	28	28	28	
25,000 - 30,000	23	24	25	26	26	27	27	27	28	28	29	29	29	30	30	
30,000 - 35,000	25	26	27	27	28	28	28	29	29	30	30	31	31	31	32	
35,000 - 40,000	26	27	28	28	29	29	30	30	30	31	32	32	32	33	33	
40,000 - 45,000	27	28	29	30	30	31	31	31	31	32	33	33	34	34	34	
45,000 - 50,000	28	29	30	31	31	32	32	32	33	34	34	34	35	35	36	
50,000 - 60,000	29	31	32	32	33	34	34	34	35	36	36	37	37	37	38	
60,000 - 70,000	31	32	34	34	35	35	36	36	36	37	38	38	39	39	40	
70,000 - 80,000	32	34	35	36	36	37	37	38	38	39	40	40	41	41	42	
80,000 - 90,000	34	35	36	37	38	38	39	39	40	41	41	42	42	43	43	
90,000 - 100,000	35	36	37	38	39	40	40	41	41	42	43	43	44	44	45	
100,000 - 125,000	37	39	40	41	42	43	43	44	44	45	46	47	47	48	48	
125,000 - 150,000	40	41	43	44	45	45	46	46	47	48	49	49	50	51	51	
150,000 - 175,000	42	44	45	46	47	48	48	49	49	50	51	52	53	53	54	
175,000 - 200,000	44	46	47	48	49	50	50	51	51	53	54	54	55	56	56	
200,000 - 250,000	47	49	51	52	53	54	54	55	55	57	58	58	59	60	61	
250,000 - 300,000	50	52	54	55	56	57	58	58	59	61	61	62	63	64	64	
300,000 - 350,000	52	55	57	58	59	60	61	62	62	64	65	65	66	67	68	
350,000 - 400,000	55	57	59	61	62	63	63	64	65	67	68	68	69	70	71	
400,000 - 450,000	57	60	62	63	64	65	66	67	67	69	70	71	72	73	74	
450,000 - 500,000	59	62	64	65	67	68	68	69	70	72	73	74	74	75	76	
500,000 - 600,000	63	66	68	69	71	72	73	73	74	76	77	78	79	80	81	
600,000 - 700,000	66	69	71	73	74	75	76	77	78	80	81	82	83	84	85	
700,000 - 800,000	69	72	74	76	78	79	80	81	81	84	85	86	87	88	89	
800,000 - 900,000	72	75	77	79	81	82	83	84	85	87	88	89	91	91	93	
900,000 - 1,000,000	74	78	80	82	84	85	86	87	88	90	92	93	94	95	96	

Table F-2. Optimum Keys/Block

Records in File	Key Size in Bytes														
	2	3	4	5	6	7	8	9	10	11 to 15	16 to 20	21 to 25	26 to 35	36 to 50	51 to 100
0 - 5,000	16	17	18	18	18	19	19	19	19	20	20	20	21	21	21
5,000 - 10,000	20	21	22	23	23	23	24	24	24	25	25	25	26	26	26
10,000 - 15,000	23	24	25	26	26	27	27	27	27	28	29	29	29	30	30
15,000 - 20,000	26	27	28	28	29	29	29	30	30	31	32	32	32	33	33
20,000 - 25,000	28	29	30	31	31	32	32	32	33	34	34	34	35	35	36
25,000 - 30,000	29	31	32	32	33	34	34	34	35	36	36	37	37	37	38
30,000 - 35,000	31	32	33	34	35	35	36	36	37	37	38	38	39	39	40
35,000 - 40,000	32	34	35	36	36	37	37	38	38	39	40	40	41	41	42
40,000 - 45,000	34	35	36	37	38	38	39	39	40	41	41	42	42	43	43
45,000 - 50,000	35	36	37	38	39	40	40	41	41	42	43	43	44	44	45
50,000 - 60,000	37	39	40	41	42	42	43	43	43	45	45	46	46	47	48
60,000 - 70,000	39	41	42	43	44	44	45	45	46	47	48	48	49	49	50
70,000 - 80,000	40	42	44	45	46	46	47	47	48	49	50	50	51	52	52
80,000 - 90,000	42	44	45	47	47	48	49	49	50	51	52	52	53	54	54
90,000 - 100,000	44	46	47	48	49	50	50	51	51	53	54	54	55	56	56
100,000 - 125,000	47	49	51	52	53	54	54	55	55	57	58	58	59	60	61
125,000 - 150,000	50	52	54	55	56	57	58	58	59	61	61	62	63	64	64
150,000 - 175,000	52	55	57	58	59	60	61	61	62	64	65	65	66	67	68
175,000 - 200,000	55	57	59	61	62	63	63	64	65	67	68	68	69	70	71
200,000 - 250,000	59	62	64	65	67	68	68	69	70	72	73	74	74	75	76
250,000 - 300,000	63	66	68	69	71	72	73	73	74	76	77	78	79	80	81
300,000 - 350,000	66	69	71	73	74	75	76	77	78	80	81	82	83	84	85
350,000 - 400,000	69	72	74	76	78	79	80	81	81	84	85	86	87	88	89
400,000 - 450,000	72	75	77	79	81	82	83	84	85	87	88	89	91	91	93
450,000 - 500,000	74	78	80	82	84	85	86	87	88	90	92	93	94	95	96
500,000 - 600,000	79	82	84	87	89	90	91	92	93	96	97	98	100	101	102
600,000 - 700,000	83	87	90	92	94	95	96	97	98	101	102	103	105	106	107
700,000 - 800,000	87	91	94	96	98	99	100	102	102	105	107	108	110	111	112
800,000 - 900,000	90	94	97	100	102	103	105	106	106	110	111	112	114	115	116
900,000 - 1,000,000	93	98	101	103	105	107	108	109	110	113	115	116	118	120	121



**Table F-3. Optimum or Minimum Index Block Size**

<p>Optimum block size = <math>10 + \left\{ \frac{(10) (OKB) (KS+4)}{9} \right\}</math></p> <p>OKB = Optimum keys/block              KS = Key size</p> <p>Minimum block size = <math>10 + \left\{ \frac{(10) (MKB) (KS+4)}{9} \right\}</math></p> <p>MKB = Minimum keys/block              KS = Key size</p> <p>NOTE: <math>\left\{ \right\}</math> = Round up if result not whole integer.</p>
--

**Table F-4. Bytes Required in Buffer for Directory-Directory Entries**

<p>Usage = <math>\left[ \frac{9(OBS-10)}{10} \right] = US</math></p> <p>Number keys/primary index block = <math>\left[ \frac{US}{KS+4} \right] = NKP</math></p> <p>Number keys/directory block = <math>\left[ \frac{US}{KS+2} \right] = NKD</math></p> <p>Total number keys represented/              directory block = <math>(NKP) (NKD) = NKRD</math></p> <p>Number entries in              Directory-directory block = <math>\left( \frac{\text{file size}}{NKRD} \right) = NKDD</math></p> <p>Number of bytes required for              buffer for directory-directory              entries = <math>10 + (KS+2) (NKDD)</math></p> <p>NOTE: <math>\left\{ \right\}</math> = Round up if result not whole integer.  <math>\left[ \right]</math> = Round down if result not whole integer.</p>
---

## CALCULATING BY FORMULA

If the user wishes to calculate keys/block based on a different file maximum than given in Tables F-1 and F-2, the following algorithms, along with Table F-5, can be used to compute minimum and optimum keys/block. The constants  $K_o$  and  $K_m$  are taken from Table F-5 based on key size.

$$\text{Optimum (OKB)} = \left\{ \sqrt[3]{\frac{FS}{K_o}} \right\}$$

$$\text{Minimum (MKB)} = \left\{ \sqrt[3]{\frac{FS}{K_m}} \right\}$$

FS = Maximum File Size

NOTE:  $\left\{ \right\}$  = Round up if result not whole integer.

Table F-5. Constants for Alternate Algorithm

KS	K <sub>o</sub>	K <sub>m</sub>	KS	K <sub>o</sub>	K <sub>m</sub>
2	1.2500	2.5000	52	.5975	1.1950
3	1.0888	2.1776	53	.5967	1.1934
4	.9877	1.9753	54	.5959	1.1918
5	.9184	1.8367	55	.5952	1.1904
6	.8681	1.7361	56	.5945	1.1890
7	.8299	1.6598	57	.5939	1.1878
8	.8000	1.6000	58	.5932	1.1864
9	.7759	1.5518	59	.5926	1.1852
10	.7562	1.5124	60	.5920	1.1840
11	.7396	1.4792	61	.5914	1.1828
12	.7256	1.4512	62	.5908	1.1816
13	.7136	1.4272	63	.5903	1.1806
14	.7031	1.4062	64	.5897	1.1794
15	.6940	1.3880	65	.5892	1.1784
16	.6859	1.3718	66	.5887	1.1774
17	.6787	1.3574	67	.5882	1.1764
18	.6722	1.3444	68	.5878	1.1756
19	.6664	1.3328	69	.5873	1.1746
20	.6612	1.3224	70	.5868	1.1736
21	.6564	1.3128	71	.5864	1.1728
22	.6520	1.3040	72	.5860	1.1720
23	.6480	1.2960	73	.5856	1.1712
24	.6443	1.2886	74	.5852	1.1704
25	.6409	1.2818	75	.5848	1.1696
26	.6378	1.2756	76	.5844	1.1688
27	.6348	1.2696	77	.5840	1.1680
28	.6321	1.2642	78	.5837	1.1674
29	.6296	1.2592	79	.5833	1.1666
30	.6272	1.2544	80	.5830	1.1660
31	.6249	1.2498	81	.5827	1.1654
32	.6228	1.2456	82	.5823	1.1646
33	.6209	1.2418	83	.5820	1.1640
34	.6190	1.2380	84	.5817	1.1634
35	.6172	1.2344	85	.5814	1.1628
36	.6156	1.2312	86	.5811	1.1622
37	.6140	1.2280	87	.5808	1.1616
38	.6125	1.2250	88	.5805	1.1610
39	.6111	1.2222	89	.5802	1.1604
40	.6097	1.2194	90	.5800	1.1600
41	.6084	1.2168	91	.5797	1.1594
42	.6072	1.2144	92	.5794	1.1588
43	.6060	1.2120	93	.5792	1.1584
44	.6049	1.2098	94	.5789	1.1578
45	.6038	1.2076	95	.5787	1.1574
46	.6028	1.2056	96	.5785	1.1570
47	.6018	1.2036	97	.5782	1.1564
48	.6009	1.2018	98	.5780	1.1560
49	.6000	1.2000	99	.5778	1.1556
50	.5991	1.1982	100	.5776	1.1552
51	.5983	1.1966			

# INDEX

ABEND	1-11;D-1	Control Language statements (continued)	
ACCEPT macro	D-2	//DECLARE	2-1,33;4-1;
Allocate routine	1-8		A-1;B-7;C-4
/* statement	2-1,41;A-2;	//DEFINE	1-8;2-1,11,15,
	B-8;C-4		28;A-1,2;
			B-3;C-1
BLOCK (BLK) parameter	2-23,27,28,	//EOT	1-1,8,11;
	28b,39;B-3,		2-1,8,10;
	6,8;C-2,3,4		A-2;B-1;C-1
BUFFER (BUF) parameter	2-21,28;	//EXECUTE	1-1,8,11;
	B-3;C-2		2-1,11;
			A-1,2;
//CALL statement	2-1,33,34;		B-1;C-1
	4-1;A-2;	//IF	1-8;2-1,31;
	B-7;C-4		A-1,2;
CATALOG (CAT) parameter	2-23,28;		B-7;C-4
	B-4;C-2	//JOB	1-1;2-1,8;
Cataloged procedures	1-6;4-1		A-1;B-1;C-1
	thru 4-4	//PAR	1-8;2-1,11,14;
CHECKOUT	3-2		A-1;B-2;C-1
Checkout debugging	E-7	//ROUTE	2-1,4,27,41;
CHECKOUT file	3-1		A-1;B-5;C-3
CLS parameter	2-37,39;	//SET	2-1,11,28d;
	B-8;C-4		A-1,2;
CODE (COD) parameter	2-31,32;		B-6;C-3
	B-7;C-4	//TELL	1-8;2-1,11,
Command field	2-4		30;B-6;C-3
Comment field	2-6	Control program	E-3
* (comment) statement	2-1,8,10;	COPY parameter	2-27,28c;
	A-2;B-1;C-1		B-6;C-3
CONTIGUOUS (CON) parameter	2-24,27,28,	CSD parameter	2-20,28;
	28b,39;B-4,6,8;		B-3;C-2
	C-2,3,4		
Continuation, statement	2-5	Data delimiter	2-41
Control flow	1-15	Data level	2-1,36;
Control Language definitions	1-1		A-2;C-4
Control Language Services	1-2;E-3	Data Management	E-3
Control Language statements	1-1;A-1,2;	//DATA statement	2-1,37;
	B-1,2;		A-1,2;
	C-1,2		B-8;C-4
/*	2-2,41;	DATE (DAT) parameter	2-28e;B-6;
	A-2;B-8;C-4		C-3
/* CLS	2-41	Debug	E-1
//CALL	2-1,33,34;	DEBUG file	1-11
	4-1;A-2;	DEBUG (DEB) parameter	2-13;B-2;
	B-7;C-4		C-1
* (comment)	2-1,8,10;	Debug program	E-4
	A-2;B-1;C-1	//DECLARE statement	2-1,33;
//DATA	2-1,37;		4-1;A-1;
	A-1,2;		B-7;C-4
	B-8;C-4	Default values	2-35

//DEFINE (DEF) statement	1-8;2-1,11, 15,28;A-1, 2;B-3;C-1	HOLD (HLD) parameter	2-10,27,28c; B-1,6; C-1,3
Device assignment	3-3	Identifier field	2-3
disc	3-5	IDENTIFIER (IDENT or ID) parameter	2-16,27,28, 28a;B-3,5; C-1,3
magnetic tape	3-3	//IF statement	1-8;2-1,31; A-1,2; B-7;C-4
telecommunication	3-5	ILOCATION (ILOC or ILO) parameter	2-25,28; B-5;C-2
unit record	3-3	Index block size	F-1
DEVICE (DEV) parameter	2-19,27,28, 28a;B-3,5;C-2, 3	Input data spooling	1-3,5
Devices	3-1	Input reader	1-3,6,7
Disc allocation	2-22;3-6	Interstep level	2-1,31; A-1;C-4
Disc device	3-5	IVOLUME (IVOL or IVO) parameter	2-25,28; B-5;C-2
Disc file expansion	2-26;3-7	JCT	1-8
Disc file organization	3-6	JDATE macro	D-1
DISPLAY macro	D-2	Job	1-1
DUMP (DMP) parameter	2-11,12; B-2;C-1	Job control table	1-8;E-1, 3,14
Dump program	E-4	Job initiator	1-3,8,9
EHALT	1-11;D-1	Job level	2-1,8; A-1;C-1
//EOJ statement	1-1,8,11; 2-1,8,10; A-2;B-1; C-1	//JOB statement	1-1;2-1,8; A-1;B-1;C-1
EXPAND (EXP) parameter	2-26,28; B-5;C-3	Job step	1-1
//EXECUTE (EXEC or EX) statement	1-1,8,11; 2-1,11; A-1,2; B-1;C-1	Job terminator	1-3,11,14
File definition	3-3	Job stream conventions	2-41
File Description Table	E-1,4	Job queuing	1-3,4a
File sharing	2-18	Keyword operand field	2-5
FILENAME (FIL) parameter	2-16,27,28, 28a,38; B-3,5,6,8; C-1,3,4	LABEL (LAB) parameter	2-20,28; B-3;C-2
Files	3-1,6	Language description	2-1
CHECKOUT	3-1	Language format	2-2
disc	3-6	Levels	2-1
SYSIN	3-1	data	2-1,36; A-2;C-4
SYSOUT	3-1,2	interstep	2-1,31; A-1;C-4
Format rotation	2-7	job	2-1,8; A-1;C-1
FORMS (FOR) parameter	2-27,28d; B-6;C-3	procedure	2-1,33; A-2;C-4
GO parameter	2-31,32; B-7;C-4	step	2-1,11; A-1;C-1
HALT	1-11;D-1		

LIBRARY (LIB) parameter	2-11,12,34, 35;B-2,7; C-1,4	Parameters (Continued)	
LIBUTIL program	4-3	DEVICE	2-19,27,28, 28a;B-3,5; C-2,3
LOCATION (LOC) parameter	2-23,28; B-3;C-2	DUMP	2-11,12; B-2;C-1
\$LODLIB	3-2	EXPAND	2-26,28; B-5;C-3
Logical I/O	3-7	FILENAME	2-16,27,28, 28a,38; B-3,5,6,8; C-1,3,4
Magnetic tape device	3-3	FORMS	2-27,28d; B-6;C-3
MEMLIM macro	D-2	GO	2-31,32; B-7;C-4
MSC parameter	2-18,28; B-3;C-2	HOLD	2-10,27,28c; B-1,6; C-1,3
NAME (NAM) parameter	2-8,9,11, 12;B-1,2; C-1	IDENTIFIER	2-16,27,28, 28a;B-3,5; C-1,3
Non-standard labeled tape	3-4	ILOCATION	2-25,28; B-5;C-2
NUMBER (NUM) parameter	2-22,27,28, 28c,39; B-3,5,6,8; C-2,3,4	IVOLUME	2-25,28; B-5;C-2
OP parameter	2-30;B-6; C-3	LABEL	2-20,28; B-3;C-2
ORGANIZATION (ORG) parameter	2-20,28; B-3,4;C-2	LIBRARY	2-11,12,34,35; B-2,7;C-1,4
Output data spooling	1-3	LOCATION	2-23,28; B-3;C-2
//PAR statement	1-8;2-1,11, 14;A-1;B-2; C-1	MSC	2-18,28; B-3;C-2
Parameters		NAME	2-8,9,11,12; B-1,2;C-1
BLOCK	2-23,27,28, 28b,39;B-3, 6,8;C-2,3,4	NUMBER	2-22,27,28, 28c,39;B-3, 5,6,8; C-2,3,4
BUFFER	2-21,28; B-3;C-2	OP	2-30;B-6;C-3
CATALOG	2-23,28; B-4;C-2	ORGANIZATION	2-20,28; B-3,4;C-2
CLS	2-37,39; B-8;C-4	PAUSE	1-8;2-30; B-6;C-3
CODE	2-31,32; B-7;C-4	PGM	2-11,12;B-1; C-1
CONTIGUOUS	2-24,27,28, 28b,39;B-4, 6,8;C-2,3,4	PRIORITY	2-9;B-1;C-1
COPY	2-27,28c; B-6;C-3	PROC	2-34;B-7;C-4
CSD	2-20,28; B-3;C-2	RESTART	2-11,13; B-2;C-1
DATE	2-28e;B-6;C-3	RETENTION	2-21,28; B-3;C-2
DEBUG	2-13;B-2;C-1	SAVE	2-27,28c; B-6;C-3

Parameters (Continued)

SIZE	2-22,27,28, 28b;B-3,5;C-2, 3	Sample jobs	G-1
SPOOL	2-27,28b; B-5;C-3	SAVE parameter	2-27,28c; B-6;C-3
SPREAD	2-24,28; B-4;C-2	Scratch files	2-17
STATUS	2-17,28; B-3;C-2	SDATE macros	D-1
SWITCH	2-27,29; B-6;C-3	//SET statement	2-1,11,28d; A-1,2; B-6;C-3
TIME	2-11,12; B-2;C-1	SETIF macro	D-2
TYPE	2-9;B-1;C-1	Sequence field	2-6
UCS	2-27,28c; B-6;C-3	Shared drive	3-6
USER	2-8,9; B-1;C-1	SIZE (SIZ) parameter	2-22,27,28, 28b;B-3,5; C-2,3
VERIFY	2-24,28; B-4;C-2	Space request	E-7
VOLUME	2-20,27,28, 28a;B-3,5; C-2,3	SPREAD (SPR) parameter	2-24,28; B-4;C-2
Partition layout	E-1,2	SPOOL (SPL) parameter	2-27,28b; B-5;C-3
Partition space pool	E-1,5,6	Spooling	
PAUSE (PAU) parameter	1-8;2-30; B-6;C-3	input data	1-3,5
Permanent files	2-18	output data	1-3
PGM parameter	2-11,12; B-1;C-1	Standard labeled tapes	3-4
Physical I/O	3-8	Standard linkage area	E-1,3,9
POST macro	D-2	Statement continuation	2-5
PRIORITY (PRI) parameter	2-9;B-1; C-1	Statement interpreter	1-6
Private load library	3-2;E-1	Statement specification	2-8
PROC (PRO) parameter	2-34;B-7; C-4	STATUS (STA) parameter	2-17,28; B-3;C-2
Procedure level	2-1,33; A-2;C-4	Step initiator	1-3,8,10
Program description	1-4	Step level	2-1,11; A-1;C-1
Program execution	1-12a	Step terminator	1-3,11,13
Relocating program loader	1-6;E-3	SWITCH (SWI) parameter	2-27,29; B-6;C-3
Replacement rules	H-1	SYSCRD	2-19,37
Required run-time variables	2-35	SYSIN file	1-8,11;3-1
RESTART parameter	2-11,13; B-2;C-1	SYSOUT file	1-8,11; 3-1,2
RETENTION (RET) parameter	2-21,28; B-3;C-2	\$\$SYSPROCLIB	4-3
//ROUTE statement	2-1,4,27,41; A-1;B-5;C-3	System control interface	D-1
RPOST macro	D-2	System input file	3-1
		System output file	3-2
		Task control table	E-1,3,11
		Telecommunications	E-8
		Telecommunication device	3-5
		//TELL statement	1-8;2-1,11, 30;B-6;C-3
		Temporary files	2-17
		TIME (TIM) parameter	2-11,12; B-2;C-1
		TYPE (TYP) parameter	2-9;B-1; C-1

UCS parameter	2-27,28c;B-6; C-3	VERIFY (VER) parameter	2-24,28; B-4;C-2
Unit record devices	3-3	VOLUME (VOL) parameter	2-20,27,28, 28a;B-3,5; C-2,3
Unlabeled tape	3-5		
Unshared drive	3-6		
USER (USE) parameter	2-8,9; B-1;C-1	Work files	2-17