IAS

User's Guide

Order No. DEC-11-OIUGA-A-D

IAS Version 1

Order additional copies as directed on the Software Information page at the back of this document.

digital equipment corporation · maynard, massachusetts

First Printing, December 1975

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-10
DECCOMM		TYPESET-11

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

CONTENTS

FIGURES

TABLES

0.2 MANUAL OBJECTIVES AND READER ASSUMPTIONS

The IAS User's Guide is intended for the person who wants to use the Program Development System (PDS) Command Language for interactive and batch processing. Part 1 is a tutorial which explains how to use most PDS commands; this part assumes no prior knowledge of the IAS system. Part 2 is a reference section, primarily consisting of a dictionary of PDS command specifications, which assumes a knowledge of the information covered in Part 1.

A copy of this guide should be immediately available to any PDS user working at a terminal.

0.2 DOCUMENT STRUCTURE

The guide consists of two parts. Part 1 is the tutorial section containing the following chapters:

Chapter	1	Introduction to IAS
Chapter	2	A Sample Interactive Session
Chapter	3	Keyboard Operation
Chapter	4	Issuing PDS Commands
Chapter	5	Batch Processing
Chapter	6	File Handling
Chapter	7	IAS Text Editors
Chapter	8	Introduction to Program Control
Chapter	9	BASIC
Chapter	10	COBOL
Chapter	11	FORTRAN
Chapter	12	MACRO-11

Part 2 is the reference section. It contains a summary description of command format and a dictionary of general PDS commands.

0.3 ASSOCIATED DOCUMENTS

Refer to the IAS Documentation Directory, Order Number DEC-11-OIDDA-A-D, for a description of all the available associated IAS documents and their readerships.

CHAPTER 1

INTRODUCTION TO IAS

1.1 TIMESHARING

IAS (Interactive Applications System) is a multifunction operating system for the PDP-11/70 and -11/45 computers. IAS supports the concurrent execution of three processing modes: Interactive, Batch and Real-time. Real-time applications operate on a priority basis, while Interactive and Batch tasks are timeshared. This manual provides a guide to Interactive and Batch processing. Real-time applications under IAS are discussed in the IAS Executive Reference Manual - Volume II.

Since timesharing participants or 'users', simultaneously require the system resources, they must be prevented from interfering with one another. Each IAS user has a User Identification Code (UIC) which the system associates with all the user's timesharing activities and files. In addition, system software and user programs and data have protection codes that determine how and if those areas may be accessed by different groups of users.

Every user identifies himself to the system by supplying a unique User Name whenever he initiates an interactive or batch job. Interactive users must also supply a password that has been previously associated with the User Name.

1.1.1 Real-time Applications

IAS provides the same real-time capabilities as DIGITAL's RSX-11D multiprogramming system. These capabilities are designed for applications that require response to physical events as they occur. Typical real-time applications include manufacturing process control, laboratory data acquisition, and communications. See the IAS Executive Reference Manual - Volume II, for further details.

1.1.2 Interactive Processing

Operating from an interactive terminal, each user may create and run programs interactively or submit them to a batch stream; alternatively, the user may exploit other users' programs or standard programs provided by the system (if his UIC grants him access). Even

though many users benefit from sharing programs and files, the system preserves the individual user's privacy and shields the activities of other users.

Interactive processing offers 2-way communication with the computer. The user initiates activities and remains in control but regulates those activities according to information that the system feeds back to him.

The interactive user communicates with the system by typing commands at the keyboard of a DECwriter or a VDU (Visual Display Unit). The standard IAS command language for the general user is PDS (Program Development System), which is described in this manual; but the user has the option of creating other command interfaces to suit particular applications.

1.1.3 Batch Processing

Once a job has been submitted to the batch stream, it can be executed without user intervention. Under IAS, programs can be created and tested interactively, and then submitted to a batch stream for execution; on the other hand, all three operations may be done in batch mode.

In batch mode, programs can be compiled or assembled, linked, and executed; devices can be claimed and released, and messages can be issued to the operator. All of these services are invoked by the same commands used for interactive processing. In interactive mode, the user can store these commands in a file which is then submitted to the batch processor. Alternatively, the batch commands may be submitted on punched cards.

Since batch requirements vary from installation to installation, and even from day to day, the IAS batch facility can be readily adjusted to meet the needs of a particular installation. For example, consider a system manager faced with a large number of daytime interactive users and a number of large batch jobs. The system manager could allocate 90% of the system's resources to interactive use during the day, and reverse the allocation at night. This would allow some batch jobs to be run during the day and some interactive jobs at night.

1.2 IAS COMMAND LANGUAGE

The standard IAS interface for interactive and batch users is provided by the Program Development System (PDS).

1.2.1 Interactive Commands

under PDS, interactive users may create, compile, link and run programs; submit jobs to a batch stream; use various peripheral devices and obtain information about the system.

PDS is a command interpreter. After PDS is activated at a terminal, either automatically or by CTRL/C (Control C), PDS invites the input of a command by issuing the prompt "PDS>". The user must provide identification to the system by logging in (entering User Name and password) before beginning interactive activity. After logging in, the user is able to make use of those IAS facilities allocated to him by the system manager (see section 1.2.3).

A typical sequence of activities during an interactive session might involve entering a source program, getting it translated into machine-executable form, and then running the program. The user requires the services of a number of system programs to do these things: an editor to enter the source program and to correct typographical and other errors; a language translator to convert the source program into object code; and, for FORTRAN and MACRO programs, the Task Builder to create an executable task.

Commands input to PDS invoke the services of these programs. PDS checks to ensure that input commands are meaningful in the current context. For example, the FORTRAN command may only be issued after a user has logged in.

1.2.2 Batch Commands

Most batch commands are identical to interactive PDS commands except that batch commands always contain a dollar sign ($) in the first position of the line, e.g. $RUN. The batch user, like the interactive user, can use PDS commands to create, compile, link and run programs, and to use various peripherals.

An interactive user may create a file of batch commands and submit the file to a batch stream; alternatively the batch job could be submitted on cards.

Interactive and batch commands are described in parallel in Part 2. The parameters of an interactive PDS command may be either prompted for, supplied as one line (with continuation characters where necessary), or issued in a combination of both methods; whereas all the parameters of a batch command must be supplied as one string, using continuation characters between lines when necessary. The command descriptions provide examples of both PDS and batch usage.

1.2.3 Restricting the Use of PDS Commands

An IAS user may discover that the system does not allow him to issue certain commands from an interactive terminal or within a batch job. This situation can occur because every manager of an IAS System determines the groups of commands that each user is allowed to issue.

For instance, the manager may decide that a certain user may only program in BASIC, and therefore allocates that user only the commands necessary for developing and running BASIC programs.

IAS allows the system manager to control the way the command language is used so that the installation's work can be carried out as efficiently as possible.

1.3 PROGRAMMING LANGUAGES

IAS supports several programming languages, including BASIC, COBOL, FORTRAN-IV and MACRO-11. The MACRO-11 Assembler is the standard IAS language, whereas translators for the other languages are optional.

Programs in BASIC and COBOL can be executed immediately after translation, because they produce intermediate code which is run by an interpreter. FORTRAN and MACRO-11 produce machine-language code and therefore require the additional step of linking.

1.3.1 BASIC

BASIC is easy to learn and use, and has found wide acceptance in educational, business, and scientific applications. PDP-11 BASIC's "immediate" mode allows each statement to be executed as it is typed in; the computer can be used like a desk calculator. Alternatively, a program can be entered, edited and then run as a unit.

1.3.2 COBOL

COBOL (COmmon Business Oriented Language) is a pseudo-English programming language designed primarily for business use. PDP-11 COBOL conforms to the American National Standard 1974 level-1 COBOL standard, with many high-level features. COBOL is an optional feature of IAS.

COBOL can be used in both batch and interactive applications. For situations where the terminal is the only input device, PDP-11 COBOL provides a simple, terminal-oriented line format. Several utility programs are provided with COBOL, including a report-generating program and a reformatting program.

COBOL can be used for both interactive and batch processing.

1.3.3 FORTRAN

The FORTRAN (FORmula TRANslation) language is especially useful in scientific and mathematical applications. PDP-11 FORTRAN conforms to the specifications of American National Standard FORTRAN (X3.9-1966), with substantial extensions to that standard.

The FORTRAN system consists of a compiler, a library of functions, and an object time system (OTS). The compiler produces object code from the source program. The OTS consists of routines that are selectively linked with the user's program to perform certain arithmetic, I/O, and system-dependent service operations. The OTS also detects and reports run-time error conditions.

There are two FORTRAN systems supported on IAS: FORTRAN-IV and FORTRAN IV-PLUS.

FORTRAN can be used for both interactive and batch processing.

1.3.4 MACRO-11

The programmer who wishes to work closely with the PDP-11 hardware and IAS may use the powerful MACRO-11 assembler. In addition to allowing the user to invoke machine-language instructions, MACRO-11 allows the programmer to define "macros" which may be invoked to generate repetitive coding sequences. The MACRO-11 language can be used both in interactive and batch processing applications.

CHAPTER 2

A SAMPLE INTERACTIVE SESSION

This chapter introduces the user to PDS by demonstrating its use in a typical session at an interactive terminal. Section 2.1 records the session, which is then described line by line in the following sections.

The line numbers at the left hand margin of the page are for reference purposes and are not part of the actual session. Underlining indicates text printed by the system.

2.1 SAMPLE SESSION

```
01          IAS PROGRAM DEVELOPMENT SYSTEM VERSION 01A  21-JUL-75
                                    17:09:08      21-AUG-75
02      PDS> LOGIN
03      USERID? CAROL
04      PASSWORD?
05      USER CAROL  UIC [200,200]  TT05:  TASK 160 17:09:21  21-AUG-75
06      PDS> CREATE ADD.FTN
07              TYPE 1
08      1       FORMAT(' ENTER TWO NUMBERS')
09              APPE\\\^R
10              ACCEPT 2,K,L
11      2       FORMAT (22\I5)
12              PRINT^U
13              TYPE 3,K+L
14      3       FORMAT(' THE SUM IS ',I5)
```

```
15              STOP
16              END
17      ^Z
18      PDS> TYPE ADD.FTN
19              TYPE 1
20      1       FORMAT (' ENTER TWO NUMBERS')
21              ACCEPT 2,K,L
22      2       FORMAT (2I5)
23              TYPE 3,K+L
24      3       FORMAT (' THE SUM IS ',I5)
25              STOP
26              END

27      PDS> FORTRAN   ADD
28      17:17:41 TASK TERMINATION
29      CORE SIZE 10K                CPU TIME 00:10

30      PDS> LINK   ADD
31      17:18:38 TASK TERMINATION
32      CORE SIZE 11K                CPU TIME 12:06

33      PDS> RUN   ADD
34      17:18:51
35      ENTER TWO NUMBERS
36      12,  78
37      THE SUM IS     90
38      JOB160 -- STOP
39      17:19:14 TASK TERMINATON
40      CORE SIZE 7K                 CPU TIME 00:00
```

A SAMPLE INTERACTIVE SESSION

```
41    PDS> DIRECTORY

42    DIRECTORY DB0:[200,200]

43    21-AUG-75 17:20

44    ADD.OBJ;1              2.              21-AUG-75 17:17
45    ADD.FTN;1              1.              21-AUG-75 17-17
46    ADD.TSK;1              32.      C      21-AUG-75 17:18
47            TOTAL OF 35. BLOCKS IN 3. FILES
48    PDS> RENAME   ADD.*;*     ADDTWO.*;*
49    PDS> DIREXCTORY/BRIEF^U
50    DIRECTORY/BRIEF
51    DIRECTORY DB0:[200,200]
52    ADDTWO.OBJ;1
53    ADDTWO.FTN;1
54    ADDTWO.TSK;1


55    PDS> LOGOUT
56    USER CAROL UIC [200,200] TT05: TASK 160 17:23:01    21-AUG-75
57    CONNECT TIME 14 M     SYSTEM UTILIZATION 12 MCTS
58    BYE
```

2.2 INVOKING PDS

The Program Development System (PDS) provides the standard IAS interface with the computer. The installation's IAS system manager determines who may use PDS and decides which terminals will support it.

Therefore, in order to issue PDS commands at a terminal, a user must be authorized to do so, and the terminal must support PDS. If these two conditions have been satisfied, then the following steps should be taken to invoke PDS:

1. Check that the terminal's power is on.
2. Set the LOCAL/REMOTE switch to REMOTE.
3. Consult installation instructions for additional required terminal settings and dial-up instructions.
4. Press CTRL/C (that is, type C while holding down the CTRL key).

The system responds to CTRL/C by displaying a PDS identifier and the current time and date. For example:

```
IAS PROGRAM DEVELOPMENT SYSTEM   VERSION ØlA 21-JUL-75

     18:29:ØØ          28-JUL-75

PDS>
```

The prompt PDS> is then displayed at the beginning of the next line to indicate that the system is ready to receive PDS commands.

In some instances the user may discover that a terminal is already prompting for PDS commands even though no one else is currently using that terminal. A user can then log into the system immediately since PDS has already been invoked.

PDS is designed to time out after several minutes (the exact number of minutes depends on the installation) if no commands have been issued and no program is running. When this happens, the system displays the messages

```
TIMEOUT

BYE
```

The user must then type CTRL/C to re-invoke PDS.

It is possible for PDS to time out while the user is typing a command; in this case the entire line must be retyped when PDS is again prompting.

2.3 PDS COMMANDS

2.3.1 The LOGIN Command

Once PDS is prompting, the user initiates an interactive session by typing

LOGIN <CR>

The symbol<CR>represents carriage return, which may be activated either by the carriage return key (CR or RETURN) or by the altmode key (ESC or ALT). One of these keys must be pressed to terminate a command string or any other line of input and to transmit the line to the system. The carriage return key and the altmode key can have different effects in certain contexts. The differences are discussed in Chapter 4, section 4.1.2.

2.3.1.1 The User Name - In response to LOGIN, PDS displays the prompt

USERID?

which asks the user to supply his User Name. The User Name is a unique 1- to 12-character alphanumeric string that identifies the individual user to the system. The system then determines the user's User Identification Code (UIC) from the User Name. The UIC determines whether the user is privileged to read or manipulate any file he attempts to access. See Chapter 6, section 6.1.3 for further details.

NOTE

The system manager assigns each user a User Name, which is then registered with IAS. A user who does not have a User Name or has forgotten it should consult the system manager.

2.3.1.2 The Password - An additional security measure to prevent unauthorized access to the system is the user's password. Once the user has entered a User Name by activating carriage return, PDS prompts

PASSWORD?

at the beginning of the next line. The user must then type in a 1- to 6-character alphanumeric string, i.e. a password, that has previously been associated with the unique User Name.

A user may change his password with the PASSWORD command (see Part 2).

Since the purpose of the password is to verify a user's identity, it should be kept secret. PDS respects the user's private password by not displaying the characters typed in after the PASSWORD? prompt.

If the password given is incorrect, PDS prompts PASSWORD? again. The user has three chances to type the password correctly before PDS exits and prints the text BYE. To begin again, the user must type CTRL/C and then LOGIN. When the user types the correct password, IAS responds by displaying the following information (line 5):

```
USER CAROL   UIC[200,200]   TT05:   TASK 160   17:09:21   21-AUG-75
```

The TASK number is assigned to the session by IAS and is normally significant only to the system manager or operator who oversees the running of the whole computer system.

The above line is followed by a new line beginning with PDS> to indicate that the system is ready to receive further commands.

2.4 THE CREATE COMMAND

After successfully logging in, the user creates a file called ADD.FTN (line 6). The CREATE command is one of several PDS commands that can be used to create a file. "ADD" is the filename and "FTN" is the file extension, which describes the contents of the file. In this case, the extension indicates that the file contains a FORTRAN source program.

After terminating the CREATE command by pressing carriage return, the user starts to enter the source program lines from the keyboard. The first typing position on each line is equivalent to position 1 on a coding sheet or punched card. The various function keys (described in Chapter 3) must be used to format the lines as required. For example, the TAB key may be used to skip 8 spaces to position the text "TYPE 1" in line 7. Carriage return terminates each line and moves the typing position to position 1 of the next line.

2.4.1 Correcting Input Errors

On line 9, the user makes a typing error, corrected by means of the DELETE key (sometimes labelled RUBOUT). The user presses the key three times to delete E, P and then P again:

```
APPE\\\
```

Each time the key is pressed, the system deletes the rightmost character on the line. Terminals with an attached printer display a backslash (\) for each deleted character. Display units actually erase each deleted character from the screen and move the printing position to the left.

In this example, the user presses CTRL/R (by typing R while the CTRL key is held down) to display the corrected text on a clean line (line 10) as follows:

```
APPE\\\^R

A
```

The user then completes the line correctly and terminates it as usual with carriage return.

```
ACCEPT 2,K,L
```

On line 11, the DELETE key is used once more to change the third 2 to I:

```
2    FORMAT(22\I5)
```

2.4.2 Cancelling a Line

By mistake the user proceeds to type "PRINT" on the next line, but then presses CTRL/U to cancel the line and start again on line 13. CTRL/U (U pressed while the CTRL key is held down) deletes a line that has not been terminated by carriage return and advances the typing position to the beginning of the next line. The user can then enter the text that was originally intended.

```
PRINT^U

TYPE 3,K+L
```

CTRL/U is a useful way to correct a line whenever it is inconvenient to use the DELETE key.

2.4.3 Closing the New File

The last statement of the source program is "END" (line 16). After entering the last statement, the user types CTRL/Z (types Z while holding down the CTRL key) to indicate to the system that the file ADD.FTN is complete. The system displays ^Z and then prompts "PDS>" on the next line.

2.5 THE TYPE COMMAND

In response to the prompt (line 18) the user issues the TYPE command to display at the terminal the file ADD.FTN as it appears after corrections. The system responds by printing the contents of the file on lines 19 through 26.

2.6 THE FORTRAN COMMAND

After checking that the source program is correct, the user decides to run it. But the program must first be translated into instructions that the computer can understand. The translated source program is an "object module" of machine instructions.

In IAS, the FORTRAN command is used to translate a FORTRAN source program. So on line 27, the user types the following:

```
FORTRAN    ADD
```

In this case the user specifies the file as ADD rather than ADD.FTN. The FORTRAN command assumes the file extension to be FTN if it is not supplied.

After translating the program, the system prints the following text on lines 28 and 29.

```
17:17:41   TASK TERMINATION

CORE SIZE 10K              CPU TIME 00:10
```

The figures "17:17:41" refer to the time at which the system finished translating the program. Line 29 shows the amount of memory and CPU time used. "00.10" indicates that the translation required less than half a second.

The system automatically places the translated FORTRAN program, now an object module, in a file named ADD.OBJ. (The file extension OBJ implies that the file contains an object module.)

2.7 THE LINK COMMAND

FORTRAN programs use a standard set of subprograms to perform certain functions. For example the FORTRAN statements TYPE and ACCEPT require the subprograms for input/output functions. The system maintains these subprograms in object module form so that they do not have to be translated each time someone uses them.

The purpose of the LINK command (line 30) in this sample session is to couple the object module contained in ADD.OBJ with the FORTRAN subprograms that it needs.

```
LINK    ADD
```

The omitted file extension is assumed to be .OBJ. If there is no file called ADD.OBJ, the system returns an error message. This might occur if a user tries to link an untranslated FORTRAN program, for instance.

Lines 31 and 32 display statistics about the completed execution of the LINK command.

The linked, executable program (the translated program linked with the required subprograms) is then placed in a file called ADD.TSK. The extension TSK stands for "task" which is IAS terminology for an executable program.

2.8 THE RUN COMMAND

The FORTRAN and LINK commands have prepared the source program for execution. The user then issues the RUN command on line 33 to activate it.

```
RUN   ADD
```

Again, the file extension may be omitted. In this case the system assumes it to be .TSK since only programs in task form can be run. Line 34 shows the time the program began to run.

The FORTRAN program ADD is interactive; it requests the user to enter two numbers, then adds them together and displays the result (lines 35 to 37)

```
ENTER TWO NUMBERS

12,   78

THE SUM IS 9Ø
```

Writers of interactive programs must remember to prompt the user. If no prompts appear, the user cannot know what data to enter or at what point to enter it. This program uses the statements on lines 19 and 2Ø to display the prompt

```
ENTER TWO NUMBERS
```

The user supplies the numbers 12 and 78 on the next line and presses carriage return to terminate the input. The program then obeys the program statements on lines 23 and 24 by adding the numbers and declaring the sum to be 9Ø. The STOP statement (line 25) then causes the program to stop and the system to print the following line:

```
JOB16Ø -- STOP
```

The job number is the number assigned to the interactive session when the user logged in (see line 5).

The information displayed on the next two lines is similar to that on lines 28, 29 and 31, 32 described in previous sections.

2.9 THE DIRECTORY COMMAND

In the session so far, the user has specifically created one file and caused the system to create two more, namely:

- ADD.FTN
- ADD.OBJ
- ADD.TSK

The system never automatically deletes a file, so all three must still exist. Only the system manager or users authorized by the file owner can delete a file.

The DIRECTORY command (line 41) causes the system to display a list of the user's existing files. File information is stored in "directories". Line 42 identifies the user's directory as [200,200].

```
DIRECTORY DB0:[200,200]
```

The first 200 identifies the user's group and the second 200 identifies the user's number within the group. The text "DB0:" indicates that the directory resides on a volume mounted on a disk drive named DB0:

Line 43 states the date and time that the listing was requested.

The next three lines list the directory information:

```
ADD.OBJ;1      2.     21-AUG-75   17:17
ADD.FTN;1      1.     21-AUG-75   17:12
ADD.TSK;1      32.  C 21-AUG-75   17:18
```

Notice that ";1" appears at the end of each file name. The number 1 is the file's version number and indicates that each file listed is the first version of the file. If the user were to issue the command FORTRAN ADD again, the FORTRAN translator would produce a second object file called ADD.OBJ;2. Users can either delete old versions or retain them as security against the loss of later versions.

The value in the second column indicates the number of 512 byte blocks occupied by each file on the disk. The date and time show when each file was created. The "C" that appears on the third line between the number of blocks and the date declares that the blocks within ADD.TSK;1 are "contiguous"; that is, they are physically located one next to the other.

2.10 THE RENAME COMMAND

The RENAME command allows the user to change the name of a file without changing its contents or location. The user now issues the command to rename all three files named ADD at the same time (line 48)

```
RENAME   ADD.*;*   ADDTWO.*;*
```

The asterisks (*) that appear in the above line are the mechanism that allow the user to specify all three files at once. An asterisk or "wild-card", is a shorthand notation for "all". ADD.*;* means all the files that have ADD as a filename, disregarding the file extension and version number. In this case, ADD.*;* refers to the files ADD.FTN;1, ADD.OBJ;1 and ADD.TSK;1. The user could also refer to these three files in the following manner:

```
ADD.*;1
```

since all the files have the same version number but different extensions.

The command issued on line 48 changes the files' name from ADD to ADDTWO. The wild-cards in the text "ADDTWO.*;*" mean that the renamed files retain their original extensions and versions. The files are now called

- ADDTWO.FTN;1
- ADDTWO.OBJ;1
- ADDTWO.TSK;1

2.11 THE DIRECTORY/BRIEF COMMAND

When the user reissues the DIRECTORY command (lines 49 and 50) the system lists the files with their new filenames. (Note that CTRL/U was pressed to cancel line 49 because of a typing error. See the description of CTRL/U in section 2.4.2).

This instance of the DIRECTORY command includes the text "/BRIEF" a "qualifier" which modifies the action of the command. /BRIEF causes the system to list only the names of the files and to omit information about blocks and time of creation.

Most commands have one or more qualifiers. A slash (/) always precedes the qualifiers name. When a user specifies more than one, the slashes sepaarate one from the next, no spaces are allowed between qualifiers or between the command name and the first qualifier.

2.1.2 THE LOGOUT COMMAND

To end the interactive session, the user issues the LOGOUT command (line 55). The system then displays user and accounting information on the next two lines and the text "BYE" on the third line.

The terminal is now inactive and CTRL/C must be pressed to invoke PDS once more.

CHAPTER 3

KEYBOARD OPERATION

The purpose of this chapter is to acquaint the user with the keyboard layouts of interactive terminals and to describe the keyboard functions and how to use them under IAS. Instructions on how to log into the system and to use PDS are contained in Chapter 4.

3.1 THE KEYBOARD

The interactive user types data directly into the system from a terminal (for example, a DECwriter or a display unit) instead of supplying input data on punched cards or paper tape. The keyboard layout of an interactive terminal is very similar to the layout of an ordinary typewriter. The number and letter keys are in the traditional typewriter format, but punctuation marks, special characters and function keys may differ in position from one type of terminal to another (see Figures 3-1 and 3-2).

3.1.1 Keyboard Functions

The user types the input text one line at a time, terminating each line with carriage return (CR or RETURN) or altmode (ALT or ESC). The system either prints the terminal input on the terminal printer or displays it on the screen of a display unit (except when the user types a password, see Chapter 2, section 2.3.1.2).

Function keys can be used to format a line (Space Bar, TAB), to edit a line (RUBOUT/DELETE), or to access the uppermost of two characters that appear on a key (SHIFT, SHIFT LOCK). The CTRL key, when pressed simultaneously with a letter key, provides further keyboard functions; these functions are described in detail in section 3.1.2. Typing a carriage return (CR or RETURN) causes the system to store the current line or to carry out some specified action.

Table 3-1 describes the function keys and the effects of their use under IAS.

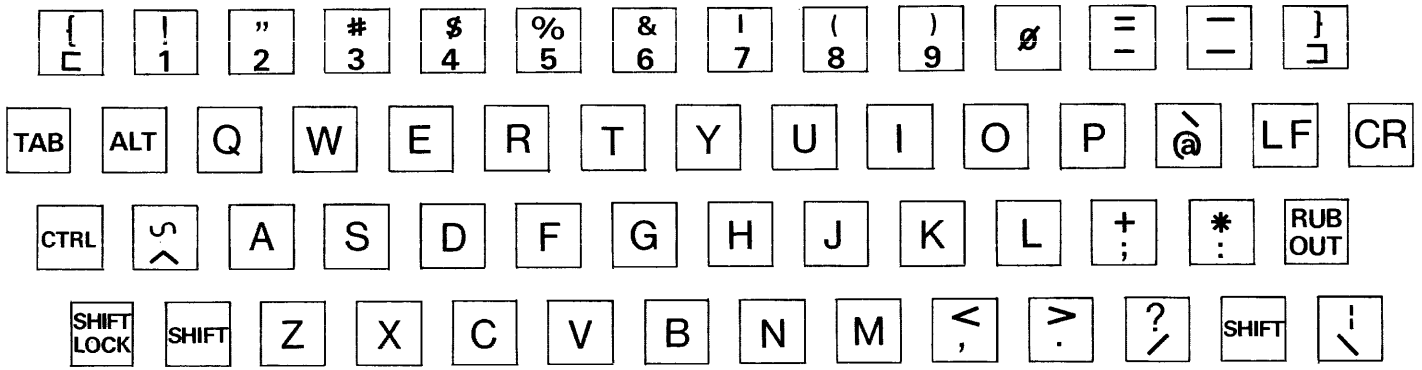

Figure 3-1
LA30/VT05 Layout

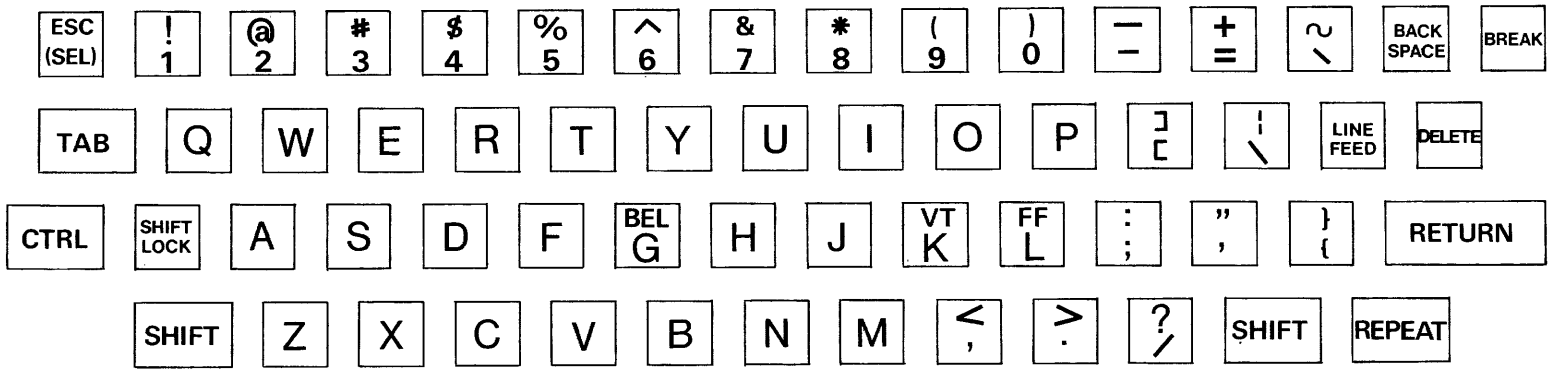

Figure 3-2
LA36/VT50 Layout

Table 3-1
Keyboard Functions

Key	Description
CR or RETURN	Carriage return. Transmits the current line to the computer and performs a carriage return. When keyed after a PDS command string, it causes PDS to suppress prompts for parameters that are optional or can be defaulted.
CTRL	Is part of several 2-key combinations that produce a variety of functions. See section 3.1.2.
DELETE RUBOUT	Deletes the last printed or displayed character or space. May be used repeatedly. On a display unit, the current printing position moves to the left and the deleted character is erased. On other terminals, the system prints a backslash (\) for each deleted character and moves the current printing position to the right. See Section 3.2
ESC or ALT	When keyed after a PDS command string, it causes PDS to prompt for the next parameter (if any).
LINE FEED or LF	Has no control effect under IAS.
SHIFT	Prints or displays the uppermost of two characters appearing on a key typed while SHIFT is held down. SHIFT has no effect when used with keys that have only one character.
SHIFT LOCK	Alternately locks and unlocks SHIFT mode.
SPACE BAR	Advances the current printing position one space at a time.
TAB	Causes the current printing position to move to the next tab stop on the line. A line conventionally contains tab stops every 8 spaces.

3.1.2 Control Key Functions

Typing a character key while pressing the control key (CTRL) invokes one of the functions listed in the following table. The combination of CTRL and another character key is called a control character. In this manual a control character is written "CTRL/X" where X is the variable character key.

The effect of a control character sometimes depends on the activity that the terminal is currently supporting.

Table 3-2 lists all the control characters supported under IAS and their associated functions.

Table 3-2
Control Key Functions

Control Character	Function
CTRL/C	Before a user has logged in, invokes PDS. Suspends the user's executing task and returns control to PDS. Cancels a command if issued between the PDS> prompt and carriage return.
CTRL/I	Causes the current printing position to move to the next tab stop on the line. Performs the same action as the TAB key.
CTRL/K	Advances the current line to the next vertical tab stop. Equivalent to a Line Feed.
CTRL/L	Advances continuous stationery to the next top of form. Equivalent to a Form Feed.
CTRL/O	Alternately suppresses and continues the printing of logical units of output on the terminal. It has no effect upon task execution.
CTRL/Q	Restarts suspended terminal output (see CTRL/S below).

Table 3-2 (Cont.)
Control Key Functions

Control Character	Function
CTRL/U	Deletes the current line. See Section 3.2.2.
CTRL/R	Retypes the current line with any deleted characters removed. See Section 3.2.2.
CTRL/S	Suspends printing of current terminal output until CTRL/Q is pressed.
CTRL/Z	Terminates a file input from a terminal, that is, signals "end of file".

3.2 CORRECTING INPUT ERRORS

Before terminating a line, the user can correct typing errors or change the line completely by using RUBOUT or DELETE or CTRL/U. However, once the line has been terminated and thus transmitted to the computer, it can be corrected only by means of an editing program.

3.2.1 Cancelling a PDS Command

Typing CTRL/C cancels a PDS command that has not yet been terminated.

3.2.2 Deleting Individual Characters

The RUBOUT or DELETE key deletes individual characters or spaces beginning with the last one entered and moving to the left. Each time the key is pressed, the system echoes a backslash (\) except when the terminal is a display unit. On a display unit, the RUBOUT or DELETE key erases the deleted characters and moves the print position to the left.

For example, to change ACCDE to ABCDE, the user presses the RUBOUT or DELETE key four times to override the characters CCDE. On any interactive terminal other than a display unit, the string then looks like this: ACCDE\\\\. The user then enters the correct sequence, BCDE. The backslashes and deleted characters are not included in the line transmitted to the computer when it is finally terminated.

On a display unit, the same correction sequence appears as follows: pressing the DELETE or RUBOUT key four times reduces the string ACCDE to A. The user then types the string BCDE to make the correct sequence ABCDE. The system displays the corrected line exactly as it is transmitted to the computer when the user presses carriage return.

3.2.3 Deleting a Line

CTRL/U deletes all characters on the line, prints ^U and performs a carriage return. The user can then enter the text correctly.

For example, if a user types ACCDEFGHI, but meant to type B for the first C, pressing the RUBOUT key eight times would be tedious and the result confusing. It would be easier to press CTRL/U and start again. The latter solution would appear as follows:

```
ACCDEFGHI ^U
ABCDEFGHI
```

After using the RUBOUT or DELETE key to correct a line and before terminating the line, the user can ensure that the final result is in fact correct. To display the line as it will be sent to the computer, simply press CTRL/R.

Further corrections can be made at this point if necessary.

3.3 USE OF UPPER AND LOWER CASE

On terminals that are equipped with upper and lower case letters, PDS commands may be entered in either case. The command is always printed in upper case no matter how it is entered. The use of lower case as input data to a program depends on the program.

CHAPTER 4

ISSUING PDS COMMANDS

4.1 COMMAND NAMES AND PARAMETERS

The user tells PDS what to do by issuing commands at an interactive terminal or by submitting commands to a batch queue. A command consists of a command name which describes the action the system is to take (COPY or LOGIN, for example), usually accompanied by one or more parameters. Parameters either describe the items on which the command is to act or further define the function of the command.

Commands can only be entered at an interactive terminal when the system is prompting "PDS>". Some PDS commands (EDIT and BASIC, for example) invoke a program that accepts its own set of commands, valid only while that program is running. In turn, PDS commands are not valid while that program is running; the user must first return control to PDS. The specifications of EDIT and BASIC in part 2 describe how to terminate the invoked program's execution.

4.1.1 Command Strings

Batch command strings contain the command name and parameters in a single or continued line. Interactive users can either supply the command name followed by the parameters on one line or enter the parameters in response to prompts (see section 4.1.3 below). In both batch and interactive mode, when two or more parameters are on one line, they must be separated by a comma, spaces and/or tabs.

If a command runs to more than one line, a hyphen (-) as the last chatacter on the line or card causes the command to be continued onto the next line.

An exclamation mark (!) after the last character of any command line indicates the start of a comment. The comment text appears after the exclamation mark. If the comment is to appear on a line that has been continued by a hyphen, the exclamation mark must immediately follow the hyphen.

4.1.2 Parameters

The parameters to the COPY command (see Chapter 6, section 6.4.2.2), which are an input file specification and an output file specification, can be input in any one of the following ways.

In interactive mode:

1. PDS> COPY RISE.MAC WORK.MAC
2. PDS> COPY RISE.MAC , WORK.MAC
3. PDS> COPY

   FROM? RISE.MAC WORK.MAC
4. PDS> COPY RISE.MAC

   TO? WORK.MAC
5. PDS> COPY

   FROM? RISE.MAC

   TO? WORK.MAC

In batch mode:

1. $COPY RISE.MAC WORK.MAC
2. $COPY RISE.MAC,WORK.MAC
3. $COPY RISE.MAC, WORK.MAC

4.1.3 Parameter Prompts

The LOGIN command demonstrates how PDS prompts for command parameters at an interactive terminal (See Chapter 2, section 2.2). The prompting facility greatly minimizes input errors by interactive users who are unsure of the command parameters.

The more experienced user may be very familiar with the commands and not need the prompts. PDS therefore suppresses prompts for parameters that are included on the previous line. For example, the LOGIN command may be input as follows:

PDS> LOGIN WILSON

PASSWORD?

Because the User Name (WILSON) was typed on the same line as LOGIN, separated from the command by a space, PDS suppresses the prompt USER-ID? and displays the next one, i.e. PASSWORD?.

NOTE

The user's password should be prompted for in order to allow the system to suppress its display.

4.1.4 Optional Parameters

Interactive PDS commands prompt for both mandatory and optional parameters. To display the prompt for an optional parameter, however, the user must use ALTmode (ESCape) rather than carriage return after the last mandatory parameter. For example:

```
PDS> MOUNT <CR>

DEVICE? DK: <CR>

VOLUME-ID? CHARLY <ALT>

LOGICAL NAME? AB
```

where LOGICAL NAME? is an optional prompt.

To suppress the prompt LOGICAL NAME?, the user must press carriage return after CHARLY. For example:

```
PDS> MOUNT DK2: CHARLY <CR>
```

NOTE

> Carriage return and ALTmode have the same effect on a command line when not used immediately before an optional prompt.

If an optional prompt has been invoked by mistake, simply press carriage return immediately after the prompt. For example:

```
PDS> MOUNT DK2: CHARLY <ALT>

LOGICAL NAME?  <CR>
```

Batch users may either omit the optional parameter from the command string if it is the last parameter, or replace the optional parameter with two commas if there are further parameters to be specified.

4.1.5 Parameter Lists

Some parameters may be replaced by a list of parameters enclosed in parentheses and separated by spaces, tabs and/or a comma. Parentheses are not required, however, when the list replaces a parameter that is the last or only parameter in the command.

Examples:

1. PDS> APPEND (FILEA.FTN,FILEB.FTN) FILEC.FTN

2. $DELETE AB.CBL;1, AB.OBJ;1

4.2 ABBREVIATED INPUT

A user only needs to enter enough of a command to distinguish it from all other PDS commands. All command names can be uniquely abbreviated to four letters.

For example, the LOGIN command may be shortened to:

LOGI

and still be accepted by the system; but LOG is not acceptable because it does not distinguish LOGIN from LOGOUT.

4.3 COMMAND AND FILE QUALIFIERS

The command string

PDS> PRINT/DELETE

is an example of the PRINT command (see Chapter 6, section 6.4.3.1). The command requests the system to output on the line printer the file specified on the next line, and to delete the file after it has been printed.

Command qualifiers modify the function of the command. The main purpose of the PRINT command is to output one or more specified files on a line printer. To delete the file or files is an option that the user indicates by specifying the command qualifier /DELETE.

For example, the qualifiers to the FORTRAN command (see Chapter 11 section 11.2), which invokes the FORTRAN compiler, determine the form of the output generated by the compiler.

Each qualifier may be abbreviated by supplying enough characters to distinguish it from any other possible qualifiers.

File specifications may also have qualifiers; these qualifiers describe properties the file has or is to have. For example, the /PROTECTION qualifier may modify the file specification supplied with the CREATE command (see Chapter 6, section 6.4.1.2). The qualifier determines the protection code applied to the newly-created file.

Example:

```
$CREATE NEWFILE.DAT/PROTECTION:(SY:RWED, OW:RWED,GR:R, WO:R)
```

4.4 UNACCEPTABLE COMMANDS OR SYNTAX

There are many reasons why PDS may not be able to execute a command.

4.4.1 Effect of Tasks Run from a Terminal

- in IAS terms, a running program is called a "task". The IAS Executive Reference Manual, Volume One describes tasks in detail.

When a task is running from an interactive terminal, the user may not issue any PDS commands until the task has terminated or been suspended. To suspend the task, the user must press CTRL/C. The user might then issue the SHOW STATUS command to check on the progress of the task. Depending on the information displayed, the user would either issue the ABORT command to abandon the task or the CONTINUE command to resume execution.

Most PDS commands cannot be issued while a task is suspended at a terminal. If the user tries to issue an unacceptable command, IAS displays the message:

```
COMMAND NOT ALLOWED     SUSPENDED TASK
```

The user must either issue ABORT to abandon the task or CONTINUE to resume it.

4.4.2 Subsystems

PDS commands are not valid when the user is operating within a subsystem such as BASIC or the Line Text Editor. The user must first return control to PDS and then issue a PDS command.

4.4.3 Error Messages

When a command fails, PDS displays an error or diagnostic message that indicates where the problem lies.

The following interactive session includes examples of command failures and the resultant system responses:

```
PDS> LOG
*COMMAND NOT UNIQUE
PDS> LOGI
```

```
 USERID?  SMITJ
*USER NAME NOT AUTHORIZED
 PDS> LOGI SMITH
 PASSWORD? (The terminal does not display the password.)
*PASSWORD?
*PASSWORD?
 USER SMITH          JOB 40 TIME 16:29:10  11-APR-75
 PDS> COPY
 FROM? A$B
*A  -  FILE NAME AND/OR EXTENSION MISSING
*$B  -  IGNORED
 PDS> DIRECTORY <ALT>
 FILE? A:B
*A  -  ILLEGAL DEVICE
*ILLEGAL FILE-SPECIFICATION
*:B -- IGNORED
```

The lines that indicate failure of one sort or another are preceded by asterisks. The reasons for their occurrence are as follows:

1. COMMAND NOT UNIQUE - The user did not type enough of the command to make it unique. The system could not tell whether LOG was a shortened form of LOGIN or LOGOUT.

2. USER NOT AUTHORIZED - The User Name (SMITJ) supplied did not grant the user access to PDS because the user had mistyped the last character.

3. PASSWORD? - By repeating the password prompt, the system indicated that the user SMITH had not typed the correct password (see Section 3.3.1.2).

4. A - FILE NAME AND/OR EXTENSION MISSING

   $B - IGNORED

   "$" is not a valid character within a file specification.

5. A - ILLEGAL DEVICE

   ILLEGAL FILE-SPECIFICATION

   :B - IGNORED

   "A" is not a valid IAS device name.

Common errors include:

- mistyping characters within a command
- not leaving a space where it is needed to distinguish between command components
- providing parameters in the incorrect order
- specifying incorrect devices

CHAPTER 5

BATCH PROCESSING

5.1 INTRODUCTION

Almost all IAS commands are applicable to both interactive and batch processing. Batch users, however, begin and end a job with the $JOB and $EOJ (End of Job) commands, rather than with LOGIN and LOGOUT (see Chapter 2). Batch commands must always begin with a dollar sign ($) in the first position of a line.

Batch users may submit a job either:

1. From an interactive terminal, or
2. Via a card reader

The first method requires the PDS command SUBMIT, which submits a file of batch commands to the batch processor. The processor queues the submitted job until all the jobs preceding it in the queue have terminated. See section 5.3 for a full description of the SUBMIT command.

When submitting a job via a card reader, the user includes the batch commands in the input stream.

For example:

```
$JOB GRAHAM CATJOB 3
$COBOL JOB.CBL
$EOJ
```

This example invokes the COBOL compiler to compile and run the source program held in the file JOB.CBL.

5.2 BEGINNING AND ENDING A BATCH JOB

The $JOB and $EOJ commands delimit a single batch job.

5.2.1 The $JOB command

The $JOB command marks the beginning of a batch job. Parameters to the command consist of, in the following order, the User Name, a job identifier and a time limit in minutes for the job's elapsed time.

For example:

```
$JOB CATHY TEST 3
```

CATHY is the User Name and TEST is the job identifier. The number 3 instructs the system to terminate the job after it has used 3 minutes of elapsed time.

The User Name is a 1- to 12-character alphanumeric string that is unique to the individual user; it is identical to the User Name parameter to the LOGIN command (see Chapter 2, section 2.3.1.1.).

The job identifier is a 1- to 6-character alphanumeric string that identifies the job.

5.2.2 The $EOJ Command

The $EOJ command terminates a batch job. It has no parameters.

5.3 THE SUBMIT COMMAND

The SUBMIT command submits a file of batch commands to a batch queue from an interactive terminal. When batch is activated to process entries from the batch queue, it begins with existing queue entries and then processes any jobs submitted while it is still active.

For example:

```
PDS> SUBMIT BATCHJOB.CMD
```

Submit the file BATCHJOB.CMD to the PDS batch processor.

See Chapter 6, section 6.4.1, for instructions on creating a file to contain the batch commands.

5.4 BATCH EDITING

IAS provides a batch-oriented editor to create and maintain source language files and data files on disk. This editor, called the Source Language Input Program (SLIPER), is described in Chapter 7.

CHAPTER 6

FILE HANDLING

6.1 INTRODUCTION

All the information that is stored in a computer system is held in logical units called files. A file is defined as an ordered collection of information. In order to store information, a source program, for instance, a user must create a file and input the source program to it.

Any subsequent attempts to access or manipulate the source program must be made in terms of the file that contains it, that is, by supplying a file specification. A file specification gives the system all the details it needs to identify the file: the device on which it is stored, the directory of the file, the file name, the extension and the version.

This chapter describes IAS file handling commands and how to use them.

6.1.1 IAS File System

The standard IAS file system for disks, DECtapes and magnetic tapes is the Files-11 system. Files-11 magnetic tapes conform to American National Standard Magnetic Tape Labels and File Structure for Information Interchange, X3.27-1969. A detailed description of the Files-11 file system is contained in the IAS Executive Reference Manual - Volume II and the IAS/RSX I/O Operations Reference Manual. Most PDS commands can only operate on Files-11 files.

6.1.2 Volumes

The magnetic media on which files are stored are called volumes, for example, disks, magnetic tapes. In order to access a file held on a volume, that volume must be mounted, that is, physically loaded on a disk or tape drive and connected to the user's task or session by the MOUNT command (see section 6.3.1). Volumes that do not hold files in Files-11 format must be mounted using the qualifier /FOREIGN.

6.1.3 Volume and File Protection

IAS protects the individual user's privacy and the system's security by providing a facility to restrict access to a volume. Magnetic tapes written in Files-11 format have a volume level protection code; that is, the protection assigned to the volume applies equally to every file within it. Disks and DECtapes, however, have both an overall protection code for access to the volume and individual protection codes for each file within it.

For the purposes of assigning protection codes, IAS defines four types of access, read (R), write (W), extend (E) and delete (D), and four categories of user, system, owner, group, world. The protection code designates the kind of access each user category is allowed. The user categories are defined as follows:

User	Description
SYSTEM:	All tasks that run under a system User Identification Code (UIC).
OWNER:	All tasks that run under the UIC of the owner of the file or volume.
GROUP:	All tasks that run under a UIC that has the same group number as the UIC of the owner of the file or volume.
WORLD:	Any task that does not fit into one of the three categories above.

The system uses the User Identification Code to determine file ownership. The system identifies a user's UIC from his User Name. The code is not necessarily unique to each user.

Volume protection is applied when the volume is initialized by the IAS system manager and can be re-specified via the MOUNT command (see the specification of MOUNT in Part 2).

A file's protection code is applied when the file is created and the code may subsequently be modified by the PROTECT command. If the user does not explicitly specify a protection code for a newly-created file, the system automatically applies the volume's default code.

Example:

```
PDS> PROTECT

FILE?  MYFILE.DAT

PROTECTION?  (SYS:RWED, WO:, G:RW)
```

The example above changes the protection code of the file MYFILE.DAT so that the system (SYS:) has all four types of access, the world (WO:) is denied all types of access, the group (G:) has read and write access, and the allowed access of the owner does not change. This example illustrates the following rules:

1. The protection code must always be enclosed in parentheses.

2. The four user categories are represented by codes followed by colons. The codes may be abbreviated to one or more letters. The codes are:

   SYSTEM:

   OWNER:

   GROUP:

   WORLD:

3. The four types of access are represented by single letters as follows:

   R Read

   W Write

   E Extend

   D Delete

4. Each category that is mentioned is allocated the types of access specified after the code and denied any type of access not specified; for example, GR:RW gives group members read and write access only.

   If no types of access are specified after a category, all types of access are denied to it, for example, WO:

5. Any category not mentioned keeps the access privileges previously allocated to it.

6. The user categories and types of access may be specified in any order.

6.2 FILE SPECIFICATIONS

A file specification provides the system with all the details it needs

- to create a file
- to identify an existing file stored on a volume
- to read a file from or write a file to a device such as a line printer or a card reader

The basic format of a file specification is as follows:

```
dev:[ufd]name.ext;ver
```

where

dev: is a device name of the form XXnn: where XX is a 2-letter mnemonic for the device (see Table 6-1) and nn is a 1- or 2-digit number, an octal number from Ø to 77.

The device mnemonics are listed in Table 6-1.

The device field may be replaced by a logical name (see Section 6.3.1).

[ufd] is the UFD (User File Directory) of the form [m,n] where m and n are octal numbers from 1 to 377.

name is the name of the file, an alphanumeric character string from 1- to 9-characters long.

ext is a 1- to 3-alphanumeric character extension that usually identifies some aspect of the file contents. Table 6-2 lists standard extensions for IAS files. For example, the extension FTN indicates that the file contains a FORTRAN source program.

ver is the version number, an octal number in the range 1 to 77777 used to differentiate among versions of the same file. For example, when a file is created, the system assigns the file a version number of 1. If that file is subsequently opened for editing, the editor keeps the original file for backup and creates a new file with the same filename and extension, but with a version number of 2.

Table 6-1 lists the 2-character mnemonics conventionally used in the device name field of file specifications.

Table 6-1
IAS Device Types

Mnemonic	Device Type
AD	AD01 A/D converter
AF	AFC11 Analog input
CO	Console output
CR	Card reader
CT	Cassette
DB	RP04 disk
DF	RF11 disk
DK	RK05 disk
DP	RP02 or RP03 disk
DS	RS03 or RS04 disk
DT	DECtape
LP	Line printer
LS	LPS A/D converter
MM	TU16 magnetic tape
MO	Message output
MT	TU10 magnetic tape
SY	User's system disk
TI	User's data input stream
TO	User's data output stream
UD	UDC11 Universal Digital Control

TI and TO are logical device names for a user's input and output data streams. For example, when a user wishes to read from his terminal he specifies TI:

```
PDS> COPY

FROM?  TI:

TO?  MYFILE.DAT
```

transfers the input text typed at the user's terminal to the file named MYFILE.DAT.

Table 6-2 lists all the standard IAS file extensions.

Table 6-2
Standard IAS File Extensions

Extension	Description
CBL	A COBOL language source file
CMD	A file containing a list of commands (an indirect file)
DAT	A data file
DIR	A directory file
FTN	A FORTRAN language source file
LST	A file in print-image format
MAC	A MACRO-11 assembly language source file
MAP	A file containing a memory allocation map
MLB	A macro library file
OBJ	An object program (output from MACRO-11 or FORTRAN)
ODL	An overlay file
OLB	An object library file
SAV	A saved system memory image file
SML	A system macro library file
SPR	A spooled output file
SRT	A sort work file
STB	A symbol table file
TMP	A temporary file
TSK	A task image file produced by the Task Builder and suitable for execution.

6.2.1 Defaults

A user may omit the device name and/or the UFD field of any file specification. In this case, the system replaces the null fields with the user's default values.

The version number may also be omitted, in which case, the system assumes:

1. The highest version number for an input file specification or
2. The highest version increased by one for an output file specification or 1 if no previous version exists.

The device and UFD defaults are determined initially as follows:

1. The default device is determined for each user by the system manager.
2. The default UFD is equivalent to the UIC (see section 6.1.3) associated with the user's User Name (submitted at log in).

The following table lists the default values, if any, of the various fields.

Table 6-3
File Specification Defaults

Field	Default
device name	At log in, the user's system device. May be changed subsequently by the SET command. The new default device must be mounted and the user must have access to it. Not to be defaulted when the file specified is to be written to or read from a record-oriented device (see section 6.2.3)
ufd	At log in, the default UFD is equivalent to the user's UIC. May be changed subsequently by the SET DEFAULT command. A user must have access to any UFD selected as a default.
name	None
extension	May be defaulted in the appropriate context. IAS has standard extensions (see Table 6-2) that it uses as defaults in defined contexts.
version	For input specifications, the highest version number. For output specifications, the highest version increased by 1 or 1 if no previous version exists.

6.2.1.1 Changing Default Values (The SET Command) - The default device or UFD used in file specifications may be changed at any time by the SET command.

To change the default device:

```
PDS> SET DEFAULT device-name
```

where device-name is the new default device.

To change the default UFD:

```
PDS> SET DEFAULT ufd
```

where ufd is the new default UFD in the format [m,n] and m and n are octal numbers between 1 and 377.

See Part 2 for a complete description of the SET command.

6.2.1.2 Displaying Default Values (The SHOW Command) - The current default values for the device field and UFD field can be displayed at an interactive terminal by using the SHOW command (see Part 2) as follows:

PDS> SHOW DEFAULT

The system responds by displaying the user's default device and ufd.

6.2.2 Wild-cards

6.2.2.1 Input Files - The user may specify more than one file in a single input file specification by using an asterisk (*) convention called a wild-card. An asterisk may be placed in any field of a specification except the device field.

The asterisk or wild-card causes the system to ignore the contents of the "wild" field and to select all the files that satisfy the remaining fields.

Examples:

DEL CATH.DAT;*	Delete all versions of the file named CATH.DAT stored on the default device and UFD.
DIR DK1:[200,200]*.LST	Display information about all the highest versions of files on DK1: in UFD [200,200] that have a LST extension.
PRINT [30,4]*.MAC;*	Print all versions of the files on the default device in UFD [30,4] that have a MAC extension.
DELETE [*,*]TONY.DAT;*	Delete all versions of the file named TONY.DAT in every directory on the user's default device.
COPY *[90,4]FORT.FOR;*	Illegal specification. The device field cannot be wild.

6.2.2.2 Output Files - When a wild-card (*) replaces a field in an output file specification, it instructs the system to replace the wild field with the corresponding field in the input file specification. The device field may not be wild.

Example:

```
PDS> COPY CATH.DAT

TO? DK2:*.*
```

Copy the highest version of the file CATH.DAT from the default device to DK2:. If no version of CATH.DAT exists in the output file UFD, the version number of the output file is 1. If the output file UFD already contains one or more versions of CATH.DAT, the newly-copied CATH.DAT is given a version number one greater than the previously highest version.

Example:

```
PDS> COPY

FROM? CATH.DAT

TO? DK2:*.*;*
```

By placing a wild-card in the version field of the output file specification the user instructs the system to retain the same version number as the input file. The system returns an error message if the output file UFD contains a file with the same name and version number as the output file.

6.2.3 Valid File Specifications

The fields of a file specification that must be supplied, depend on the type of file being described. There are two types of file:

1. Retrievable files written to or stored on disks, DECtapes or magnetic tapes. These files are called named files because they have file names that the system can access.

2. Files that are read from or written to record-oriented devices (for example, a card reader or a line printer) or files held on unlabelled tapes. These files are called unnamed files.

The filename field of a named file must always be supplied; that is, the user must give an alphanumeric filename or a wild-card(*). Many commands have a default value for the extension field. However, with any command that has no default; the file extension field of a named file must always be supplied. The device, UFD and version fields may be omitted because they do have default values (see section 6.2.1). The device field may also be replaced by a logical name (see section 6.3.1).

The use of wild-cards in a file specification depends on the IAS command with which it is issued. Where it is relevant, the command descriptions in Part 2 describe restrictions on the use of wild-cards.

The specification of any unnamed file, a file read from or written to a record-oriented device, consists only of the device field, which may be a specific device or a logical name (see section 6.3.1). If any

other field is supplied, it is ignored by the system because UFDs, file names, extensions and versions have significance only for named files. The device field may not be wild.

6.3 DEVICE MANAGEMENT

Before a batch or interactive user can access a device, the device must be available. In other words, the device must be attached to the system and, in the case of a removable volume, the volume must be physically loaded. Also, if the device is nonsharable, no-one else must be using it. For example, if all tape drives are already in use, the system cannot grant a new request for a tape drive.

If the conditions are such that a device is available, the user then gains access to the device by "allocating" it, that is, by issuing a command that requests the system's permission to use it (see section 6.3.2). An exception to this procedure occurs when the user wants to access a system device.

6.3.1 System Devices

A system device is a device allocated to all users by the system manager. For example, the user's system disk, the line printer and the card reader are normally system devices.

A device such as a line printer cannot be shared by two user's simultaneously, but many users may want to access it at the same time. The system manager may therefore choose to adopt a technique called spooling. In the case of a line printer, spooling causes all output written to the printer to be queued. The system then creates disk files of all line printer output, maintains a queue containing a list of these files and prints them one at a time.

6.3.2 Accessing a Device

In order to use a non-system device, three mechanisms are required:

1. A means of obtaining access to the device (the MOUNT and ALLOCATE commands).

2. A means of keeping commands, especially in batch mode, independent of a particular physical device (Logical Device Names).

3. A means of keeping the Input/Output statements in a program independent of a particular physical device (Logical Unit Numbers).

Access to a non-system device is obtained by issuing the ALLOCATE and/or the MOUNT command. Some devices, such as disk drives, are shareable. Thus a user may mount a disk even though it has already been mounted by another user. The volume is physically unloaded when the last user to access it dismounts it.

A user is granted exclusive access to non-sharable devices.

Note that access to any volume is subject to the normal protection restrictions (see section 6.1.3).

6.3.2.1 Logical Device Names - IAS uses logical names to permit the commands written by a user to be independent of a particular physical device. If, for example, an installation has two tape drives called MTØ: and MT1:, specifying MTØ: in batch commands or indirect command files would prevent the user from using the other tape drive without changing the commands. The user may define a logical device name, TA:, for example, and use it in place of the corresponding physical device name in all subsequent commands.

Once an equivalence has been established between a logical device name and a physical device name, the logical device name may be used in any command. If a logical device name is the same as a physical device name, IAS assumes that the reference is to the logical device name.

Logical device names may be defined in ALLOCATE or MOUNT commands. A logical name has the syntax:

```
XX[nn]:
```

where XX represents two alphabetic characters and nn is an optional unit number, an octal number ranging from Ø to 77. If nn is omitted Ø is assumed.

6.3.2.2 Logical Units - All program Input/Output (I/O) is performed on logical units, which are identified by numbers (logical unit numbers or luns). Before a logical unit can be used for I/O, a physical device or file must be assigned to it. Since different devices or files may be assigned to the logical units on successive runs of a program, the program itself can be device-independent.

Users may assign logical units in three ways:

1. By using a LINK option during task build.
2. By issuing an ASSIGN command.
3. By establishing the assignment within the program before the file in question is accessed.

The LINK option and the ASSIGN command may be used to assign a physical or logical device to a logical unit. From within a program, however, the user may assign a named file to a logical unit. See one of the following manuals for further details:

1. The IAS Executive Reference Manual - Volume II
2. The appropriate IAS FORTRAN User's Guide
3. The PDP-11 COBOL Language Reference Manual
4. The BASIC-11 Language Reference Manual
5. The IAS/RSX-11 MACRO-11 Reference Manual

6.3.3 The MOUNT command

In order to access a file held on magnetic media, the volume on which it is held must be physically loaded and mounted. System devices are already mounted for and allocated to every user. For all other volumes, however, the user must issue a MOUNT command to make the device available and gain access to the volume residing on it.

Example:

```
PDS> MOUNT

DEVICE? DK2:

VOLUME-ID? TESTER
```

The command above mounts the volume labelled "TESTER" on DK2:. The user can now access any file on the mounted volume, as long as the file's protection code permits the attempted access.

The unit number in the device specification may be omitted if the user does not know or care on which unit the volume is to be mounted. If the unit number has been omitted in batch mode, the user must then supply a logical name for the device; the logical name replaces the device name in subsequent file specifications. In interactive mode, the system displays a message giving the unit on which the volume was actually loaded.

Example:

```
$MOUNT DK: TESTER DRØ:
```

The user assigns the logical name DRØ: to the unknown unit. The logical name can now be used instead of the physical device name in subsequent commands.

Files-11 disks and DECtapes are shareable volumes which can be mounted and accessed by more than one user. Magnetic tape, however, can only be mounted and accessed by one user at a time.

The system considers any volume not in Files-11 format to be "foreign". A foreign volume can only be mounted by one user at a time and the system must be told that it is foreign.

Example

```
PDS> MOUNT/FOREIGN

DEVICE?  DTØ:

VOLUME-ID?  TESTER
```

The command qualifier /FOREIGN tells the system that TESTER is not in Files-11 format and prevents other users from mounting it.

If the foreign volume is in DIGITAL's DOS or RT-11 format, file qualifiers to the COPY, DELETE and DIRECTORY commands allow the user to access files held on the volume. Otherwise, most PDS commands do not apply to foreign files.

See the specification of the MOUNT command in Part 2 for further details.

6.3.4 The DISMOUNT Command

When a user has finished accessing a volume, the DISMOUNT command should be issued in order to dismount the device and make it available for other users.

The DISMOUNT command automatically deallocates the device unless the user specifies the qualifier /KEEP. See the command specification in Part 2 for further details.

Examples:

1.
```
PDS> DISMOUNT

DEVICE?  DKØ:

VOLUMEID?  TESTER
```

2.
```
$DISMOUNT DTØ: TAPEA
```

The parameters to DISMOUNT are the device specification or logical name of the device to be dismounted and the volume identification.

6.3.5 The ALLOCATE Command

If a device is not a system device and it cannot be mounted, the ALLOCATE command must be used to access it.

Example:

```
PDS> ALLOCATE

RESOURCE?  DEVICE

DEVICE?  LP1:
```

The above example allocates a line printer to the user. No one else can use the printer until the user who allocated it issues a DEALLOCATE command (see section 6.3.5).

The ALLOCATE command may also be used to obtain exclusive access to a shareable device.

Example:

```
$ALLOCATE DEVICE DK:  MCØ:

$MOUNT MCØ VOL1
```

In the above example, a batch user has allocated a DK type disk drive and assigned it the logical name MCØ:. No one else is allowed to access that drive until is has been deallocated.

Once a device has been allocated, several volumes may be mounted one after the other.

For example:

```
$ALLOCATE DEVICE DK:  DV1:

$MOUNT    DV1:   VOL1

   .

   .

   .

$DISMOUNT/KEEP DV1:

$MOUNT DV1:   VOL2

   .

   .

   .

$DISMOUNT DV1:
```

In this example, the user obtains exclusive access to a disk drive via the ALLOCATE command. A volume labelled VOL1 is then mounted on the drive. When the user dismounts VOL1, the /KEEP qualifier retains the user's exclusive access to the disk. When VOL2 is dismounted, however, the disk is deallocated since the user does not specify /KEEP.

6.3.6 The DEALLOCATE Command

After issuing an ALLOCATE command to obtain exclusive use of a non-mountable device (a line printer or card reader, for example), a user must issue the DEALLOCATE command to free the device.

Example:

```
$ALLOCATE DEVICE LP1:
.
.
.
$DEALLOCATE DEVICE LP1:
```

The DISMOUNT command automatically deallocates an allocated mountable device unless the user specifies the /KEEP qualifier.

Example:

```
$ALLOCATE DEVICE DK:  MCØ:
$MOUNT MCØ: CATH

     .

     .

     .

$DISMOUNT/KEEP MCØ:

$DEALLOCATE DEVICE MCØ:
```

6.3.7 The ASSIGN Command

The ASSIGN command is used to associate a logical or physical device with a logical unit. (See section 6.3.2 for a definition of logical devices and logical units.)

Example:

```
PDS> ASSIGN

FILE?  LPØ:

LUN?  6
```

This command assigns LPØ: to the logical unit 6. If a program writes to logical unit 6 via The FORTRAN statement WRITE (6..., for example, the results of the write will be printed on the line printer.

6.4 FILE MANAGEMENT

6.4.1 Creating Files

6.4.1.1 User File Directories - To create a file on a volume, the volume must be mounted (see section 6.3) and the user must have write access to a User File Directory (UFD) on the volume. A UFD is a file that contains details of all the files that have been created on that volume under the UFD identifier (i.e. [m,n] where m and n are octal numbers from 1 to 377).

Interactive users can issue the DIRECTORY command to display the contents of a User File Directory at the terminal. In batch mode, the directory information is sent to the user's output stream (TO).

Example:

```
PDS> DIRECTORY


DIRECTORY DB0:[200,200]

21-AUG-75 17:20

ADD.OBJ;1          2.              21-AUG-75 17:17
ADD.FTN;1          1.              21-AUG-75 17:17
ADD.TSK;1          32.       C     21-AUG-75 17:18

        TOTAL OF 35. BLOCKS IN 3. FILES
```

If no parameter is supplied, the system displays information about the user's current default UFD. However, by supplying one or more file specifications the user can interrogate other directories or specific files.

Example:

```
PDS> DIRECTORY ADD.OBJ


DIRECTORY DB0:[200,200]

21-AUG-75 17:20

ADD.OBJ;1          2.              21-AUG-75 17:17

        TOTAL OF 17. BLOCKS IN 1. FILE
```

To interrogate DOS or RT-11 files, modify the file specification with the /DOS or /RT11 file qualifier.

Example:

```
PDS> DIRECTORY

FILE?  RTFILE.MAC/RT11
```

A User File Directory is like any other file; it has a protection code which determines who has access to it. A user may therefore create a file under any UFD to which he has write access.

6.4.1.2 The CREATE Command - Both batch and interactive users may create files by using the IAS command CREATE.

The interactive user types CREATE and supplies a file specification (no wild-cards allowed), optionally modified by the /PROTECTION qualifier. If the /PROTECTION qualifier is not specifically supplied, the new file is assigned the default file protection associated with the volume.

For example:

```
PDS> CREATE

FILE? FORT.FTN/PRO:(OW:RWED SY:  GR:  WO:)
```

The system uses default values (see section 6.2.1) for the device, UFD and version fields.

Once the command string has been terminated, the user types input to the new file, line by line.

When terminated, each line is sent to the file exactly as it has been formatted at the terminal. The user then closes the file by typing CTRL/Z.

The batch user supplies the command name optionally modified by /DOLLARS and a file specification (no wild-cards allowed), optionally modified by the /PROTECTION qualifier. The qualifier /DOLLARS tells the system that the file will be closed by the $EOD command. Otherwise, any $ (i.e. batch) command terminates the file. Therefore, the /DOLLARS qualifier must be specified whenever a record in the file being created contains a $ in position 1.

Examples:

1.
```
$CREATE/DOLLARS FORTRAN.FTN/PRO:(OW:RWED SY:  GR:  WO:)
        .
        .
        .
$EOD
```

2. $CREATE DK2:[30,4]CALCULATE.MAC

6.4.1.3 Using the Editor to Create a File - Users can also create files by means of the EDIT command. See Chapter 7 for a description of the IAS text editors.

6.4.2 Manipulating Files

This section describes how to use various IAS commands to manipulate existing files in the following ways:

- To append one or more files to an output file
- To copy a file
- To rename an existing file

6.4.2.1 The APPEND Command - The APPEND command may be used to add one or more files onto the end of an existing file.

Examples:

1. PDS> APPEND (A.CBL, B.CBL)

   TO? C.CBL

   Append files A.CBL and B.CBL to the end of the file C.CBL.

2. $APPEND MYFILE.MAC YOURFILE.MAC

   Append MYFILE.MAC to the end of YOURFILE.MAC.

NOTE

A user must have extend access to a file before appending to it.

The user specifies the input file or files (enclosed in parentheses if more than one) first and then the output file.

Input files may be retrieved from a mounted volume, input from a record-oriented device (for example, a card reader) or typed in from an interactive terminal. When more than one input file is supplied, the system appends the files in the order in which they are specified.

If one of the files is to be input from the user's terminal (TI), the system transfers to the input file everything typed at the terminal after the command string until the user types CTRL/Z to close the file.

Example:

1. $APPEND (FILE1.MAC, FILE2.MAC), FILE3.MAC

   The system adds the input files FILE1.MAC and FILE2.MAC to the file FILE3.MAC.

2. PDS> APPEND

   FILE? JUD.CBL

   TO? GRAVES.CBL

   The file JUD.CBL is appended to the output file GRAVES.CBL.

6.4.2.2 The COPY Command - The COPY command creates a duplicate of the contents of an input file in a specified output file. Optional command qualifiers allow the output file to be modified in various ways.

Examples:

1. PDS> COPY

   FROM? MT2:FRED.MAC

   TO? DK2:JIM.MAC

2. $COPY MT2:FRED.MAC, DK2:JIM.MAC

The examples above copy the highest version of FRED.MAC on MT2: to DK2: and change the file name to JIM on DK2:. As well as copying from one device to another, the COPY command can be used to copy a file from one User File Directory to another.

Example:

1. PDS> COPY [30,4]FRED.MAC

   TO? [100,100]FRED.MAC

This example copies the file FRED.MAC in [30,4] to UFD [100,100]; the filename remains unchanged.

Four of the possible command qualifiers are:

/ALLOCATION:n

/CONTIGUOUS

/OWN

/REPLACE

These are explained in detail in Part 2, but some examples of their use are shown below:

1. $COPY/ALLOCATION:20 DK2:OLDFILE.DAT DK0:OLDFILE.DAT

   Copy OLDFILE.ONE from DK2: to DK0: and make the output file 20 blocks long. The /ALLOCATION qualifier is useful for copying a contiguous file and changing its size.

2. PDS> COPY/CONTIGUOUS

   FROM? MT2:TU71.MAC DK1:*.*

   Copy TU71.MAC from MT2: to DK1: and make the output file contiguous. The wild-cards (*) indicate that the fields of the output specification in which they occur take the corresponding field values of the input file specification.

3. $COPY/REPLACE MT1:SAME.OBJ;4 DK2:SAME.OBJ;4

   The /REPLACE qualifier indicates that the output file overrides a file in the user's default UFD that has the same name, extension and version number. That is, if a file called SAME.OBJ;4 already exists on DK2: in the default UFD, it is deleted and replaced by the new one copied from MT1:

There are two file qualifiers available with the COPY command, /RT11 and /DOS, that allow the user to copy files to or from an RT-11 or DOS formatted volume. The qualifier must modify the specification of the file currently in DIGITAL'S DOS or RT-11 format. DOS and RT-11 files cannot be renamed within IAS; therefore, the filename and extension fields of the output file specification must always be wild.

Examples:

1. PDS> COPY

   FROM? DK2:FRED.DAT/RT11

   TO? *.*

   Copy the RT-11 file FRED.DAT from the foreign volume on DK2: to the user's default device and UFD. The /RT11 qualifier instructs the system to translate the RT-11 file into Files-11 format.

2. $COPY TEST.MAC;8 DT0:*.*/DOS

   Copy the Files-11 file TEST.MAC;8 to a DOS-formatted foreign volume held on DT0:.

6.4.2.3 Renaming Files - The RENAME command may be used at any time to change the name of a file. The examples below change the file name DEBUG.MAC;1 to RUN.MAC;1.

1. $RENAME DEBUG.MAC;1 RUN.MAC;1

2. PDS> RENAME

   OLD? DEBUG.MAC;1

   NEW? RUN.MAC;1

6.4.3 Listing Files

Files may be listed on a line printer or at the user's terminal. One of the commands discussed below should be used; the choice is dependent on the kind of listing desired and whether the user is operating in interactive or batch mode.

6.4.3.1 Listing on the Line Printer - The PRINT command may be used to print files on the line printer. The system often queues all line printer output until all output previously submitted to the queue has been processed. The output files are normally printed in the order in which they were submitted to the queue.

The PRINT command is the simplest way to submit a file to the line printer. For example:

1. PDS> PRINT

   FILE? FILE1.DAT, FILE2.DAT, FILE3.DAT

2. $PRINT LIST.MAP

The file or files to be printed are specified after the command.

The PRINT command provides the option to delete files after they have been printed. The user indicates this option by supplying the command qualifier /DELETE. For example:

```
$PRINT/DELETE MYFILE.DAT
```

6.4.3.2 Listing Files at an Interactive Terminal - The TYPE command causes one or more specified files to be printed at the user's interactive terminal.

Examples:

1. PDS> TYPE

   FILE? FIRST.MAC, SECOND.MAC

2. PDS> TYPE TYPE.CBL

6.4.3.3 The DUMP Facility - The DUMP command lists a specified file on the user's terminal (TO) or sends the listing to a specified output file. Command qualifiers modify the form of the listing. For example, the user may specify that the file be dumped in ASCII mode. The DUMP facility is useful for debugging programs and for displaying nonprintable characters in ASCII or octal format. See the full specification of DUMP in Part 2 for all the available options.

Examples:

1. PDS> DUMP/ASCII

   FILE? DUMP.CBL

   List the file DUMP.CBL in ASCII format on the user's terminal.

2. $DUMP/BYTE/OUTPUT:DK2:DISKFILE.OBJ OBJECT.DAT

   Send a listing of the file OBJECT.DAT in byte octal format to a file named DISKFILE.DAT on DK2:

3. PDS> DUMP/OUT:LPØ: FILE.DAT

   List the file FILE.DAT in word octal format (the default) on the line printer.

6.4.4 Deleting Files

The DELETE command deletes files held on Files-11 disks or DECtapes, or DIGITAL's RT-11 or DOS files held on foreign disks or DECtapes.

Specifications of DOS or RT-11 files must be modified by a file qualifier, either /DOS or /RT11 as appropriate.

Wild-cards(*) (see section 6.2.2) are allowed in the file specification. If the version field is omitted or wild, the command qualifier /KEEP:n may be supplied to preserve the highest n versions of the file or files specified.

Examples:

1. PDS> DELETE/KEEP:2

   FILE? MATRIX.DAT;*

   Delete all but the last 2 versions of the file MATRIX.DAT

2. $DELETE ROW.OBJ;4 COLUMN.MAC;4 PEEK.*;*

   Delete all files named PEEK and the fourth version of the files ROW.OBJ and COLUMN.MAC.

3. PDS> DELETE DK2:DOSFILE.DAT;4/DOS

   Delete the file DK2:DOSFILE.DAT;4, which is in DIGITAL's DOS format.

The PRINT command modified by the /DELETE qualifier can be used to delete files that have been submitted to the line printer. See section 6.4.3.1.

6.4.5 Summary of File Handling Commands

Command	Function
ALLOCATE $ALLOCATE	Allocate a specified device to the user.
APPEND $APPEND	Add one or more files to the end of a specified file.
ASSIGN $ASSIGN	Assign a device to a logical unit.
COPY $COPY	Copy an input file to a specified output file.
CREATE $CREATE	Create a file as specified. File contents to be input from an interactive terminal, or, in batch, to follow immediately after the $CREATE command.
DEALLOCATE $DEALLOCATE	Deallocate a specified device.
DEASSIGN $DEASSIGN	Deassign a device from a logical unit.
DELETE $DELETE	Delete specified Files-11, DIGITAL's DOS or RT-11 formatted files.
DISMOUNT $DISMOUNT	Dismount a specified volume.
DUMP $DUMP	List the contents of a file on a line printer.
EDIT $EDIT	Edit an existing file or create a new file.
MOUNT $MOUNT	Make a volume available to the user.
PRINT $PRINT	Print one or more files on the line printer.
RENAME $RENAME	Change the filename of an existing file.
PROTECT $PROTECT	Assign a specified protection code to a file.
TYPE	List a file at the user's interactive terminal.

CHAPTER 7

IAS TEXT EDITORS

This chapter provides the user with the basic information needed to run either of the two IAS editors:

- The Text Editor (EDI), primarily for interactive use, and
- The Source Language Input Program and Editor (SLIPER), a batch-oriented editor.

The IAS Editing Utilities Reference Manual contains a complete description of both editors.

7.1 THE TEXT EDITOR

The EDIT command automatically invokes the Text Editor, also known as EDI, unless the qualifier /SLIPER has been specified. EDI is an interactive context-editing program that uses editor commands to create and modify source programs and other files containing ASCII data. The specification of the EDIT Command in Part 2 contains a complete list of editor commands. This section introduces some basic editing concepts and describes a useful subset of commands.

Editor commands, as in PDS, describe the action to be performed. Each command consists of a command name followed by a single parameter. Most command names can be abbreviated to 1, 2 or 3 letters. Some command names, however, are themselves abbreviations of their function and cannot be abbreviated further. For example, NP which stands for Next Print, has no alternative form.

7.1.1 Editing Modes

EDI operates in two modes: input mode and edit mode. In input mode, EDI considers all lines entered at the terminal to be input to the file. This mode is used to create a file and to insert lines of text into an existing file. In edit mode, EDI treats lines entered at the terminal as editor commands intended to modify or manipulate existing text.

7.1.2 Input Mode

7.1.2.1. Creating a New File - if the user specifies a non-existent file with the EDIT Command, EDI automatically creates a new file and enters input mode. The specification of the file must include an extension. For example:

```
PDS> EDIT NEWFILE.DAT
[EDI -- CREATING NEW FILE]
INPUT
```

The user then begins to enter text on the next line. All characters typed are written to the file. The function and CTRL characters are used to format the lines of text (see Chapter 3).

To enter a blank line into the text, type one or more spaces at the beginning of a new line, followed by carriage return.

7.1.2.2 The INSERT Command - If EDI is operating in edit mode, the editor command INSERT, immediately followed by carriage return, changes the operating mode to input.

7.1.2.3 Changing to Edit Mode - To switch from input to edit mode, type carriage return as the first character in a line. EDI responds by displaying an asterisk (*) on the next line. The asterisk is the EDI prompt for editor commands.

7.1.3 Edit Mode

The asterisk (*) prompt indicates that EDI is operating in edit mode, and is therefore only accepting editor commands.

7.1.3.1 Editing an Existing File - To edit an existing file, supply the specification of the file with the EDIT command. The specification must include an extension. If the version number is omitted, EDI selects the highest version of the file. EDI then retrieves the input file and prompts for an editor command. For example:

```
PDS> EDIT
FILE? OLDFILE.DAT
[PAGE    1]
*
```

7.1.3.2 Block Editing - By default, EDI accesses a file in 80-line blocks, called pages. (This chapter discusses only this method of

access to the file; the alternative method, called line-by-line mode, is described in the IAS Editing Utilities Reference Manual.) The editor command SIZE may be used to change the number of lines per page (see the specification of the EDIT command in Part 2).

7.1.3.3 The Line Pointer - EDI is a context editor; it locates the line to be edited by means of text contained within the line, rather than by sequence numbers, as does the batch editor SLIPER, described in section 7.2. EDI uses a line pointer to indicate the current line to be edited.

When edit mode is first entered, the line pointer points to a line immediately preceding the first line of text in the file. The user then moves the line pointer by searching for a particular piece of text or by using commands that reposition the pointer.

For example:

```
PDS> EDIT NEWFILE.DAT
[EDI -- CREATING NEW FILE]
INPUT
THIS IS LINE 1 ENTERED
HERE IS LINE 2
LINE 3
LINE 4 WHICH IS ALSO THE LAST LINE
<CR>
*TOP
[PAGE    1]
*LOCATE LINE 1
THIS IS LINE 1 ENTERED
*NEXT
*PRINT
HERE IS LINE 2
*LOCATE ALSO
LINE 4 WHICH IS ALSO THE LAST LINE
*LOCATE ENTERED
[EDI -- EXIT]
```

In this example, the user has created a file consisting of 4 lines. When the prompt for editor commands (*) appears, the user issues the TOF (Top of File) command to move the line pointer to the top of the file. "Top" means the line immediately preceding the first line of text. The user then types "LOCATE LINE 1" to find the first line that contains the character string "LINE 1". EDI moves the pointer to that line and prints it. The LOCATE command always searches down the file beginning at the line immediately following the current line.

The NEXT command is used to advance the line pointer to the next line, which is "HERE IS LINE 2". The PRINT command then causes EDI to display the new current line without moving the pointer. "LOCATE ALSO" causes the line pointer to be moved to the fourth line, which is then printed.

The command "LOCATE ENTERED" causes the editor to print "[EDI -- *EOB*]". "EOB" is the abbreviation for End Of Buffer. Since the line pointer only moves down the text when searching for character

strings, it encounters the end of the buffer without finding the string "ENTERED." the TOF command could then be used to reposition the line pointer at the beginning of the text.

7.1.4 Editor Commands

This section describes a useful subset of EDI commands. The complete set is listed in the specification of EDIT in Part 2. The IAS Editing Utilities Reference Manual specifies all the commands in detail.

The subset of commands is described in alphabetical order. Brackets ([and]) indicate that the enclosed value is optional.

Note that the function keys carriage return (CR or RETURN) and ALTmode (ALT and ESC) can be used as editor commands. See the description of the NP Command, section 7.1.4.8. CTRL/Z may also be used to close the editing session and return control to PDS; but it is advisable to use the editor command EXIT for this purpose.

The subset includes:

Commands	Sections
CHANGE	(section 7.1.4.1)
DELETE	(section 7.1.4.2)
EXIT	(section 7.1.4.3)
FIND	(section 7.1.4.4)
INSERT	(section 7.1.4.5)
LOCATE	(section 7.1.4.6)
NEXT	(section 7.1.4.7)
NP	(section 8.1.4.8)
PRINT	(section 7.1.4.9)
PLOCATE	(section 7.1.4.10)
RENEW	(section 7.1.4.11)
RETYPE	(section 7.1.4.12)
TOF	(section 7.1.4.13)

7.1.4.1 The CHANGE Command

Format

```
[n]CHANGE /string-1/string-2[/]
```

where string is a character string. The slashes (/) delimit each string, and are therefore called delimiters. The delimiters may be any matching characters that do not appear in either string. The first character following the command is considered to be the first delimiter. The closing delimiter is optional.

n is a positive integer.

The command name may be abbreviated to one or more letters.

Function

This command searches for string-1 in the current line and, if found, replaces it with string-2. If string-1 is given but EDI cannot locate the string in the current line, EDI prints "NO MATCH" and returns an * prompt. The command can be reentered using the correct string construct. If string-1 is null (not given), string-2 is inserted at the beginning of the line. If string-2 is null, string-1 is deleted from the current line. The search for string-1 begins at the beginning of the current line and proceeds across the line until a match is found. If string-1 occurs more than once on the current line, only the first occurrence is changed.

A numeric value n preceding the command results in the first "n" occurrences of string-1 being changed to string-2. For each replacement of string-1 with string-2, the entire line is rescanned beginning at the first character in the line. This allows the user to generate a string of n characters as shown in the example below.

If no match occurs, a NO MATCH message is displayed.

The Line Pointer

The CHANGE command does not change the position of the line pointer.

Examples

1. The current line reads "333". The following command changes it to "C33":

```
*C/3/C/
```

2. The current line reads "DIAGNOSIS". The following command changes it to read "DIAGNOSTICS":

```
*CHA "IS"TICS"
```

3. The current line contains "A;B;C;D". The following command changes it to read "A;;;;;B;C;D:

```
*4C/;/;;
```

7.1.4.2 The DELETE Command

Format

```
DELETE [n]
```

where n is a positive or negative integer.

The command name may be abbreviated to one or more letters.

Function

This command causes lines of text to be deleted in the following manner:

1. If n is positive, the current line and n-1 lines following the current line are deleted. The line-pointer advances to the line following the last deleted line.

2. If n is negative, the current line is not deleted, but the specified number of lines that precede it are deleted. The line pointer remains unchanged.

3. If n is omitted, the current line is deleted and the line pointer advances to the next line.

The Line Pointer

See items 1, 2 and 3 in the Function section above for the command's effect on the line pointer.

Examples

To delete the previous five lines in the block buffer, type the following command:

```
*D -5
```

7.1.4.3 The EXIT Command

Format

```
EXIT
```

The command name may be abbreviated to two or more letters.

Function

This command transfers all remaining lines in the block buffer and input file (in that order) into the output file, closes the file and causes EDI to exit. The system then prompts for PDS commands.

Example:

```
*EX
[EDI -- EXIT]


PDS>
```

7.1.4.4 The FIND Command

Format

```
[n]FIND [string]
```

where n is a positive integer and string is a character string that begins in the first position of a line. The command name may be abbreviated to one or more letters.

Function

This command searches the block, beginning at the line following the current line, for string, which must begin in column one of the lines searched. If string is not specified, the line pointer simply advances one line. If n is given, EDI searches for the nth occurrence of string and positions the line pointer at the line that contains it.

FIND is useful for locating FORTRAN statement numbers and MACRO-11 statement labels.

The Line Pointer

If string is not given, the line pointer advances one line.

If string is given, the line pointer moves to the first or nth line containing string.

Example

```
*F LOOK
LOOK AT THE FIRST CHARACTER IN THE LINE
```

The above command causes EDI to search the block for a line beginning with LOOK and to print the line when it is found.

7.1.4.5 The INSERT Command

Format

```
INSERT [string]
```

where string is a character string.

The command name may be abbreviated to one or more letters.

Function

This command inserts string immediately following the current line. If string is omitted, EDI enters input mode.

The Line Pointer

The line pointer moves to the line in which string is inserted, that is, the line following the current line.

Example

*I TEXT INSERT IN EDIT MODE	Inserts a line of text immediately after the current line.
*F ABC	Finds a line beginning with ABC.
ABC IS THE START OF THE ALPHABET	This is the line found.
*I	An I followed by a carriage
TEXT INSERT 1 IN INPUT MODE	return causes EDI to switch
TEXT INSERT 2 IN INPUT MODE	to the input mode and a
ETC.	series of new lines can be input following the current line.

7.1.4.6 The LOCATE Command

Format

[n]LOCATE [string]

where n is a positive integer and string is a character string.

The command name may be abbreviated to 1 or more letters.

Function

This command causes a search of the buffer beginning at the line following the current line for string, which may occur anywhere in the line sought. If string is not specified, the line following the current line is considered a match. A numeric value n preceding the command results in locating the nth occurence of string. EDI then prints the located line.

The Line Pointer

EDI moves the line pointer to the line containing string or the nth occurence of string.

Example

See section 7.1.3.3.

7.1.4.7 The NEXT Command

Format

NEXT [n]

where n is a positive or negative integer.

The command name may be abbreviated to one or more letters.

Function

If n is omitted, this command causes the line pointer to advance one line.

If n is supplied, the line pointer moves forward n lines if n is positive, or back n lines if n is negative.

Example

The following command moves the current line pointer back five lines:

```
*N -5
```

7.1.4.8 The NP (Next Print) Command

Format

```
NP [n]
```

where n is a positive or negative integer.

The command cannot be abbreviated.

Function

This command has the same function as the NEXT command (see section 7.1.4.7) except that it prints out the new current line.

Note that pressing carriage return (CR or RETURN) performs the same function as NP 1, and pressing ALTmode (ALT or ESC) performs the same function as NP -1.

Example

The following four lines are contained in the file and the line pointer is at the first line.

```
LINE 1 OF THE FILE
LINE 2 OF THE FILE
LINE 3 OF THE FILE
LINE 4 OF THE FILE
```

If the following command is issued, EDI would return the following printout

```
*NP 2
LINE 3 OF THE FILE
```

7.1.4.9 The PRINT Command

Format

```
PRINT [n]
```

where n is a positive integer.

The command name may be abbreviated to one or more letters.

Function

This command prints out the current line and the next n-1 lines on the terminal. If n is omitted, the command prints the current line.

The Line Pointer

The line pointer is positioned at the last line printed if n is given. If n is omitted, the line pointer does not move.

Example:

```
*P3             Prints out the current line then the next two
                lines.
LINE 1
LINE 2
LINE 3
*P              Prints the current line only (the last line
                printed by the previous command).
LINE 3
```

7.1.4.10 The PLOCATE (Page Locate) Command

Format

```
[n]PLOCATE [string]
```

where n is a positive integer and string is a character string.

The command name may be abbreviated to two or more letters.

Function

This command searches for string in the current block and successive blocks, starting from the line following the current line. String may be positioned anywhere in the line in which it occurs. If n is specified, EDI searches for the nth occurrence of string. The line containing string is then printed.

If string is omitted, the line pointer advances one line.

The Line Pointer

The line pointer advances to the line containing string (or the nth occurrence of string) or to the line following the current line if string is omitted.

Example

The following command locates the line "HAPPY DAYS ARE HERE AGAIN", which occurs somewhere in the file ahead of the current line.

```
*PL PPY
HAPPY DAYS ARE HERE AGAIN
```

7.1.4.11 The RENEW Command

Format

```
RENEW [n]
```

where n is a positive integer.

The command name may be abbreviated to three or more letters.

Function

If n is omitted, the command writes the current block into the output file and reads a new block into the buffer. If n is specified, EDI reads n-1 blocks into the buffer and then writes them to the output file. The nth block is then read into the buffer and the line pointer positioned at the top of it.

Example

```
*RENEW 1Ø
```

In this example, ten consecutive blocks are transferred from the input file to the block buffer. Only nine blocks, however, are transferred to the output file. The current line pointer is pointing to the first line in the tenth block which is currently in the block buffer.

7.1.4.12 The RETYPE Command

Format

```
RETYPE [string]
```

where string is a character string.

The command name may be abbreviated to one or more letters.

Function

This command replaces the current line with string. If string is omitted, the command deletes the current line.

The Line Pointer

The line pointer does not move.

Example

```
*RETY THIS IS A NEW LINE
```

In this example, the string "THIS IS A NEW LINE" replaces the current line.

7.1.4.13 The TOF (Top Of File) Command

Format

```
TOF
```

The command cannot be abbreviated.

Function

This command returns the line pointer to the top of the input file and saves all blocks (pages) previously edited. The "top" of the file is the line that immediately precedes the first line of text in the file.

Example

```
*TOF
```

This command causes the previously edited pages to be written into the output file. The line pointer then moves back to the top of the file.

7.1.5 Error Messages

Refer to the IAS Editing Utilities Reference Manual for a list of EDI error messages and recommended responses.

7.2 BATCH EDITING

The Source Language Input Program and Editor (SLIPER) is a batch-oriented editing program used to create and maintain source language files on disk. It permits the user to:

1. Edit an existing source file. Commands are provided to:
   a. Delete
   b. Replace
   c. Insert
2. Obtain line number listings of files.

SLIPER accepts input from

1. The input stream
2. Any Files-11 volume, i.e. a disk or DECtape in IAS format (see Chapter 6, section 6.1.1)

Before starting SLIPER, the user should be aware of the following restrictions.

1. Before editing a file, the user must know the numbers of the lines to be edited. A current line-number listing must therefore be at hand.
2. The batch editor does not accept input lines greater than 80 ASCII characters in length. If more than 80 characters are specified, an error is declared.
3. Line numbers to which the edit commands refer must be in ascending sequence throughout the file. Form feeds and page directives are treated as part of the text.

7.2.1 Invoking SLIPER

To invoke SLIPER the user must issue the EDIT command modified by the command qualifier /SLIPER. For example:

```
$EDIT/SLIPER OLDFILE.MAC
```

Further EDIT command qualifiers applicable only to SLIPER determine the format of the output files.

Table 7-1 lists the SLIPER qualifiers and their effects.

Table 7-1
SLIPER Qualifiers

Qualifier	Description	Default
/OUTPUT[:filespec]	Produce an output file. Unless filespec is specified, the file is given the same name as the input file, with a version number increased by 1.	/OUTPUT
/NOOUTPUT	Do not produce an output file.	
/LIST[:filespec]	If /OUTPUT has been specified print a listing of the output file on the line printer. If /NOOUTPUT has been specified, print a listing of the input file on the line printer. If filespec has been specified, name and store a listing file accordingly.	/LIST
/AUDIT	Enable the editing audit trail, which indicates the changes made during the most recent editing session.	/AUDIT
/NOAUDIT	Disable the editing audit trail.	
/BLANK	Insert blanks at the end of the text line (rather than tabs) to right-justify the audit trail text.	/BLANK
/NOBLANK	Do not insert blanks at the end of the text line.	
/DOUBLE	Produce a double-spaced listing file.	/NODOUBLE
/NODOUBLE	Produce a single-spaced listing file.	

7.2.1.1 Obtaining a Listing - Note that to produce a listing of the file to be edited, the user must specify the /NOOUTPUT qualifier. For example:

```
PDS> EDIT/SLI/NOOUTPUT/LIST CHARLES.MAC
```

The command above prints a listing of CHARLES.MAC on the line printer. In batch, the default is /LIST, but interactive users must specify that qualifier to obtain a listing. The listing provides the line numbers to be used in subsequent editing of CHARLES.MAC.

7.2.2 SLIPER Output Files

When a file is edited, SLIPER produces an output file on disk under the name specified by the user. If the /AUDIT qualifier is specified (default condition), the file contains an audit trail, indicating changes effected by the editing session.

Each line that has been inserted during the last editing session is flagged by appending the characters ;**NEW** to the line.

The line following the inserted line(s) may be flagged by the characters ;**-n, where n is a decimal value equal to the number of lines that were deleted from the old file. For example:

```
;THIS IS A NEW LINE ADDED TO THE FILE ;**NEW**
;THIS IS THE NEXT LINE                ;**-1
```

indicates that the new line has simply replaced one of the old lines; that is, the edit command looked like:

```
;THIS IS A NEW LINE ADDED TO THE FILE
-m, m
```

where m is the number of the line that was replaced. There may also be entries of the following kind:

```
;THIS LINE IS A REPLACEMENT ;**NEW**
;NEXT OLD LINE              ;**-16
```

indicating that a new line has been inserted, but 16 lines have been deleted immediately preceding the next old line.

Lines may also be flagged with the characters ;**N, with no preceding new lines, to indicate that lines have been deleted without being replaced.

If /AUDIT has been specified, the current flags are stripped before the updated file is output; thus, the flags are reliable indicators of the most recent update of the file.

7.2.3 SLIPER Edit Commands

Following the initial command line, the user enters text lines, or deletes or corrects lines in the original source file. Text that is to be inserted at the beginning of the file is entered immediately following the initial command line. To correct or replace one or more lines, or to insert text in the middle or at the end of the file, the user must first specify an edit command in line position 1, followed by a decimal value that refers to a line in the input file. For example:

```
-9
```

The minus sign and line number may appear as the only element on the line, or they may be followed by a comma and a second line number, as:

```
-9,12
```

or

```
-9,9
```

SLIPER interprets the user's purpose by examining the edit command. When a single line number is specified (e.g. -9 alone), SLIPER interprets the user's purpose to be the insertion of new text lines into the source file. The line number indicates that the new text is to be inserted following the specified line (in the first example, new text would be placed in the file following line 9).

When the user provides an edit command in the second format (-9,-12), SLIPER deletes all text lines from line 9 through line 12, inclusively. The user can follow the edit command with lines of text, which will be inserted into the file in the location previously occupied by the deleted lines (that is, the first new line is the new line 9).

The edit command (-9,9) indicates that SLIPER is to delete line 9. If a text line (or lines) follows. It replaces the deleted line.

NOTE

> Line numbers must always be specified in ascending sequence. Thus, -9,8 is illegal, and an error message is printed. It is also illegal to refer to a line number lower than a line number that was referred to in a prior edit command.

7.2.3.1 SLIPER Edit Control Characters - SLIPER recognises four characters as edit control characters when they appear in line position 1:

The minus sign (-)

The "less than" sign (<)

The slash (/)

The "at" sign (@)

Table 7-2 describes their use as edit control characters.

Table 7-2
SLIPER Edit Control Characters

Character	Function
-(minus)	Indicates that an editing function is to be performed, with reference to the line number(s) specified. -n Insert text following line n. -n,n Delete line n. -n,m Delete lines n through m inclusively (m must be greater than n).
/(slash)	The slash is placed in the first position of a line to indicate that the editing of a file is completed.
@(at)	The @ character is put in the first location of a line to indicate that SLIPER is to seek input from an indirect file. The user must specify the indirect file immediately after the @ sign; for example: @DK2:DKSFIL.CMD instructs SLIPER to read input from the file DKSFIL.CMD on physical device unit DK2:. Indirect files are more fully described in Section 7.2.4. Unless otherwise specified, the file extension defaults to .CMD.
<(less than)	The < character is used when entering a line that begins with one of the special edit control characters. It causes the line to be shifted one character to the left, with the result that the < is deleted, and the desired control character is entered into the file as the first character on the line.

7.2.4 Indirect Files

Indirect files can be used to contain both editing commands and correction lines to be inserted into the file being edited. (See Chapter 8, Section 8.2.)

7.2.5 SLIPER Editing Examples

The following examples show the various editing functions that SLIPER can perform, and the command formats used.

EXAMPLE A

```
$EDIT/SLI JONES.MAC
-23,23
R1=SIZE OF BLOCK TO ALLOCATE IN BYTES
-33
     MOV   $FRHD,R2 GET ADDRESS OF FREE POOL HEADER
-36,36
-39,39
     ASR     R1      ;CONVERT TO WORDS
/
```

This example performs the following editing functions:

- Line 23 is replaced by a corrected version (i.e.; R1 = SIZE OF BLOCK TO ALLOCATE IN BYTES.);
- A new line is inserted after line 33;
- Line 36 is deleted (and not replaced);
- Line 39 is replaced by a corrected version (i.e., ASR R1 CONVERT TO WORDS),

EXAMPLE B

```
$EDI/SLI CATHS
-55,55
       BCS    60$    ;IF CS YES
-107,107
       CALL    $ERMSG ;OUTPUT ERROR MESSAGE
/
```

Example B performs the following editing functions:

- Line 55 is replaced by a corrected line;
- Line 107 is replaced by a corrected line.

EXAMPLE C

```
$EDI/SLIP/OUTPUT:CHAS.MAC   CATHS.MAC
-15,16
CNTRL:    .BYTE     '9,'Ø
-33,35
$CDTD::   MOVB      #'9,CNTRL    ;SET DECIMAL LIMIT
-38,38
COTB::    MOVB      #'Ø,CNTRL    ;SET OCTAL LIMIT
-43,45
          CMPB      #' ,R        ;BLANK?
          BEQ       1$           ;IF EQUAL YES
          CMPB      #HT,R5       ;HT?
          BEQ       1$           ;IF EQUAL YES
-47,5Ø
3$:       MOV       R5,R2        ;SET TERMINAL CHARACTER
/
```

Example C performs the following editing functions:

- Lines 15 and 16 are deleted and replaced by a corrected line;
- Lines 33 through 35 are deleted and replaced by the line starting with $CDTD;
- Line 38 is replaced;
- Lines 43 through 45 are replaced by four text lines:
- Lines 47 through 5Ø are deleted;
- Line beginning with 3$: is inserted.
- The output file is created under the name CHAS.MAC.

CHAPTER 8

INTRODUCTION TO PROGRAM CONTROL

IAS supports several programming languages, including BASIC, COBOL, FORTRAN and MACRO-11. MACRO-11 is a standard feature of IAS; the other language translators are optional. This chapter is an introduction to some language-independent aspects of running programs under IAS. The next four chapters, one on each language, describe how to use IAS commands to transform source programs into executing programs or tasks.

8.1 PROCESSING MODES

Whether it is better to operate in batch or in interactive mode depends on the nature of the programmer's job and the requirements of the installation. Interactive mode is convenient for complicated editing of source programs, for instance, or the execution of programs that require small amounts of input data. On the other hand, batch processing is usually the best mode for processing large amounts of data, for example, a payroll or accounts receivable.

8.2 INDIRECT FILES

An indirect file is a sequential file containing command input. For example, rather than repeatedly typing commonly used command sequences, the user can type the sequence once and store it in a file. To execute the sequence, the user issues an "at" sign (@) followed by the file specification instead of the first command in the sequence. The indirect file may be invoked from any position within the command string, but any characters that follow the indirect file specification are ignored. The system then retrieves the indirect file and executes the commands contained therein.

Example:

```
PDS> EDIT FILE.CMD

[ EDI  --  CREATING NEW FILE]

INPUT

FORTRAN/OBJECT/LIST:CPROG   CPROG
```

```
LINK CPROG

RUN CPROG

<CR>

*EXIT

PDS>

.
.
.

PDS> @FILE
```

The indirect file called FILE.CMD, created by means of the Line Text Editor, contains commands to compile, link and run the source program CPROG.FTN.

These commands are executed when the user invokes the file by typing @FILE in response to the PDS prompt. CMD is the default extension for indirect files in both interactive and batch mode.

In a batch context, the same command sequence could be created and invoked in the following manner:

```
$CREATE/DOLLARS FILE.CMD

$FORTRAN/OBJECT/LIST:CPROG CPROG

$LINK CPROG

$RUN CPROG

$EOD
.
.
.

@FILE
```

Note that the $CREATE command string must include the qualifier /DOLLARS, so that the system recognizes the following text as input and not as further batch commands to be processed. The $EOD command terminates the file to be created.

The command file may be invoked subsequently by the command line @FILE. No dollar sign ($) is needed.

Both batch and interactive users may invoke indirect files on up to three levels. An indirect file can itself invoke another indirect file; the second file may invoke a third; but the third file may not invoke a fourth indirect file.

See Chapter 6, section 6.4.1 for a description of file creation.

8.3 USER LIBRARIES

The IAS command LIBRARIAN allows users to create and maintain their own libraries of commonly-used macros (macro libraries) and routines (object module libraries).

8.3.1 Macro Libraries

MACRO-11 macros may be held in source (text) form in a macro library. Each macro is identified by its macro name. To use one or more macros contained in a macro library file, the programmer must supply the library file specification, modified by the qualifier /LIBRARY, in the list of input files to the MACRO command. (see the description of the MACRO command in Part 2.) The macro library must be specified before the module that calls it.

8.3.2 Object Module Libraries

Commonly-used routines are stored in object (that is, compiled or assembled) code which the user can then incorporate in a task. The object code routines are called object modules; the files in which they are held are called object module libraries.

A programmer invokes an object module from within the program, then specifies the library containing the module by means of the /LIBRARY qualifier to the LINK command (see Chapter 11, section 11.3). The Task Builder automatically searches all system libraries; but it only searches user-written libraries that have been explicitly specified in the command.

The IAS Task Builder Reference Manual describes object module libraries in detail.

The specification of the LIBRARIAN command in Part 2 describes how to create and maintain the libraries.

8.4 CREATING SOURCE FILES

Either the CREATE command or the EDIT command may be used to create source files. The EDIT command has the advantage that it allows the user immediate access to all its editing facilities. To correct errors made while using the CREATE command, however, the user must rely on keyboard facilities or close the file and then issue the EDIT command.

8.4.1 The CREATE Command

To create a source file with the CREATE command, the user must do one of two things:

1. In batch mode, issue the command $CREATE, optionally modified by the qualifier /DOLLARS, followed by a file specification of the file to be created. Insert the source program immediately after the command name. The source file is terminated either by another batch command or, if /DOLLARS has been specified, by the command $EOD.

2. At an interactive terminal, issue the PDS command CREATE followed by the file specification of the file to be created. Begin to input the source program at the beginning of the next line. Close the file by typing CTRL/Z.

The CREATE command is described in greater detail in Chapter 6, section 6.4.1.1.

Examples:

1. $CREATE/DOLLARS COBOL.CBL

```
.
.
.
00078      IF NF-DELIMITER = CR
00079         PERFORM READ-TRAN-LINE
00080         IF EOFFOUND GO TO G5999
00081         ELSE GO TO GS5
00082      IF CHAR-COUNT ZERO
00083         IF INMARKER < TRAN-LINE-LIMIT GO TO G25.

.
$EOD
```

2. PDS> CREATE

```
FILE?  TEST.FTN
SUBROUTINE PROCI
C     FIRST DATA PROCESSING ROUTINE
C     COMMUNICATION REGION
COMMON/DTA/A(200),I
.
.
.
RETURN
END
CTRL/Z
```

8.4.2 The EDIT Command

The EDIT command allows the interactive user both to create and edit a source file via the Text Editor. Batch users should use the CREATE command to create a source file, which may be edited subsequently in either interactive or batch mode (See Chapter 7).

When the EDIT command specifies a non-existent file, the Line Text Editor creates one and prompts for input. For example:

```
PDS> EDIT
FILE?  NEWSOURCE.CBL
[EDI -- CREATING NEW FILE]
INPUT
```

The user then begins to enter the source file beginning at the first position of the next line after 'INPUT'.

See Chapter 7 for details on how to use the Text Editor to edit the new file as it is being created.

To close the new file, the user must type carriage return as the first character in the line. This action causes the Editor to display an asterisk (*), which indicates that it expects an editor command rather THAN further input to the file because the command mode has changed from insert to edit. To close the file and exit to PDS, use the command EXIT. If the user wants to create further files, the EDIT command must be reissued.

Example:

```
PDS> EDIT
FILE?  TONY.FTN
[EDI -- CREATING NEW FILE]
INPUT
      SUBROUTINE REPORT
C     INTERIM REPORT PROGRAM
C     COMMUNICATION REGION
      COMMON/DTA/A(200),
      RETURN
      END
<CR>
*EX
[EXIT]
PDS>
```

CHAPTER 9

BASIC

9.1 INTRODUCTION

BASIC-11 provides immediate translation and storage of a user program while it is being input from an interactive terminal. The PDS user invokes the BASIC interpreter by typing the command BASIC. The BASIC system may not be used in batch mode under IAS.

The interactive nature of BASIC removes the need for separate steps in the development of a program. Once BASIC has been invoked, a program may be created, translated and run in a single session.

This chapter describes how to invoke BASIC, create and execute a program and then terminate a session. The following manuals describe the BASIC language itself:

BASIC-11 Language Reference Manual

IAS BASIC User's Guide

9.2 THE BASIC COMMAND

When the user issues the BASIC Command, BASIC displays as follows:

```
PDS> BASIC
READY
```

The text 'READY' indicates that BASIC is ready to receive a command or program line.

The BASIC command has no parameters or command qualifiers.

9.3 CTRL/C

If the user presses CTRL/C while a BASIC program is running, the system stops execution after the current line and dislays the number of the last line executed. The user may then issue further BASIC commands.

CTRL/C typed during the execution of a BASIC LIST or SAVE command or an immediate mode statement stops the execution of those commands or statements. It has no effect on the execution of other BASIC commands.

9.4 TERMINATING A BASIC SESSION

To terminate a BASIC session and return control to PDS, the user must type 'BYE' on a new line. The system then prints information about the session and prompts for further PDS commands. For example:

```
BYE


15.57.32 TASK TERMINATION


PDS>
```

9.5 EXAMPLE

```
PDS> BASIC
READY
OLD MYBASIC
LISTNH
10 REM PROGRAM TO TRANSLATE MONTH NAMES TO NUMBERS
50 T$ = "JANFEBMARAPRMAYJUNJULAUGSEPOCTNOVDEC"
100 PRINT "TYPE THE FIRST 3 LETTERS OF A MONTH";
110 INPUT M$
120 IF LEN (M$) <> 3 GO TO 200
130 M=(POS(T$,M$,1) + 2) /3
140 REM CHECK IF MONTH IS SPELLED CORRECTLY
150 IF M <> INT (M) GO TO 200
160 PRINT M$" IS MONTH NUMBER"M
170 GO TO 100
200 PRINT "BAD MONTH" GO TO 100
READY

RUNNH
TYPE THE FIRST 3 LETTERS OF A MONTH? NOV
NOV IS MONTH NUMBER 11
TYPE THE FIRST 3 LETTERS OF A MONTH? DEC
DEC IS MONTH NUMBER 12
TYPE THE FIRST 3 LETTERS OF A MONTH? JAN
JAN IS MONTH NUMBER 1
TYPE THE FIRST 3 LETTERS OF A MONTH? AUD
BAD MONTH
TYPE THE FIRST 3 LETTERS OF A MONTH?  CTRL/C
STOP AT LINE 110
READY
BYE
12.39.27 TASK TERMINATION
PDS>
```

In this example the user first invokes BASIC by issuing the BASIC command. BASIC indicates that it is ready to accept BASIC program lines and commands by printing READY. The user then retrieves an existing BASIC program by entering the OLD command. This program is printed and executed by the LIST and RUN commands respectively. Since this program is written as a loop, that is, after executing line 200 it loops back to line 100, it will execute indefinitely. By entering CTRL/C the user terminates the program execution. BASIC then prints the number of the line at which execution was stopped. The BYE command terminates the BASIC session.

PART 1

TUTORIAL

CHAPTER 10

COBOL

To create an executable COBOL program, the programmer needs to create a source file and submit the source to the COBOL compiler. When the program has been successfully compiled, the programmer uses the RUN command to execute it. The object program produced by the COBOL compiler runs within the framework of the COBOL system. A COBOL program cannot be linked to programs written in other languages.

This chapter describes how to use IAS commands to create source files and to compile and run COBOL programs. See Chapter 5 for a description of the SUBMIT command, which allows the user to create a file of IAS commands to be submitted to a batch stream. Consult the following manuals for information about programming in COBOL on PDP-11 machines:

PDP-11 COBOL Language Reference Manual

PDP-11 COBOL User's Guide

10.1 CREATING SOURCE FILES

Either the CREATE command or the EDIT command may be used to create source files. See Chapter 8, section 8.3. The EDIT command has the advantage that it allows the interactive user immediate access to all its editing facilities. To correct errors made while using the CREATE command, however, the user must rely on keyboard facilities or close the file and then issue the EDIT command.

10.2 THE COBOL COMMAND

By default, the COBOL command compiles a source program, produces an object file and. in batch mode, produces a listing file. For example:

```
PDS> COBOL

FILE? SOURCE.CBL
```

This command string compiles the program SOURCE.CBL. and produces an object file named SOURCE.OBJ. If the user omits the extension field in

the specification of the source file, the COBOL complier assumes it to be CBL.

10.2.1 Command Qualifiers

The qualifiers to the COBOL command are:

```
/OBJECT[:filespec]
/NOOBJECT
/LIST[:filespec]
/NOLIST
/SWITCHES:(switches)
```

The compiler produces an object file unless the user specifies /NOOBJECT. The object file may be named by default or given a name specified after /OBJECT. See the command specification in Part 2 for further details.

In batch mode, the compiler automatically produces a listing file unless the user specifies /NOLIST. Interactive users must specify /LIST to obtain a listing. The /LIST:filespec qualifier also allows the user to send the listing to a file; otherwise, the listing file is printed at the line printer and then deleted.

The qualifier /SWITCHES is described in the next section.

10.2.2 Compiler Switches

The PDP-11 COBOL compiler provides switches to tailor compilation to particular needs. The user specifies the switches by means of the /SWITCHES qualifier to the COBOL command. For example:

```
$COB/SWITCHES:(/MAP)      SOURCE.CBL
```

The switch /MAP tells the compiler to produce a Data Division map. The compiler automatically prints a listing file.

The specified switches must be enclosed in parentheses. For example:

```
PDS> COBOL/SWITCHES(/ERR:2/MAP/CVF)/LIST:ACCOUNT.LST

FILE? ACCTS.CBL
```

When the user does not specify any switches, the compiler operates according to defaults. The default switches are:

```
(/ERR:0/ACC:1/NOMAP)
```

The COBOL command specification in Part 2 defines all the possible compiler switches.

10.3 RUNNING A COBOL PROGRAM

To run a compiled COBOL program, issue the RUN command and specify the program file modified by the /COBOL qualifier.

Example:

```
$RUN TEST/COBOL
```

If the extension is omitted, the system assumes it to be OBJ.

10.4 DIAGNOSTIC ERROR MESSAGES

The compiler generates diagnostic error messages whenever it detects an error in the source program. With some exceptions, a source error detected by the compiler results in the associated diagnostic message being embedded within the source program listing. That is, when an error is detected in the source program, the compiler prints the diagnostic message either before or after the erroneous source program statement.

See the PDP-11 COBOL User's Guide for a detailed description of diagnostic error messages.

CHAPTER 11

FORTRAN

A FORTRAN programmer must complete four steps to transform a FORTRAN source program into an executing task:

1. Create one or more source files,
2. Compile the source files,
3. Link the compiled, i.e. object, files, and
4. Run the executable task.

This chapter describes how to use IAS commands to perform these steps.

See Chapter 5 for a description of the SUBMIT command, which allows the user to submit a file of IAS commands to a batch stream. A user could create such a file to compile, link and run his task in a single batch job.

Consult the following manuals for information about programming in FORTRAN IV or FORTRAN IV-PLUS:

IAS/RSX-11 FORTRAN IV User's Guide
FORTRAN IV-PLUS User's Guide
PDP-11 FORTRAN Language Reference Manual

11.1 CREATING SOURCE FILES

Either the CREATE command or the EDIT command may be used to create source files. See Chapter 8, section 8.3. The EDIT command has the advantage that it allows the user, immediate access to all its editing facilities. To correct errors made while using the CREATE command, however, the user must rely on keyboard facilities or close the file and then issue the EDIT command.

11.2 THE FORTRAN COMMAND

The basic function of the FORTRAN command is to compile one or more FORTRAN source programs. Command qualifiers, including compiler switches and options, determine the form of the output to be generated by the compiler.

11.2.1 Compiling Source Files

Only one source file may be specified with each FORTRAN command. The following command strings all compile the source file INVERT.FTN.

1. PDS> FORTRAN

   FILE? INVERT

2. $FORTRAN INVERT

3. PDS> FORTRAN INVERT

Each of the command strings above instructs the system to compile the source file specified and to produce compiler output as the defaults dictate.

By default, the compiler:

1. Produces an object file which is given the name of the source file and the extension OBJ.

2. In batch mode, produces a listing file on the line printer. No listing is produced in interactive mode unless the user requests it.

3. Compiles the source file according to the compiler's default switches. (See the FORTRAN command specification in Part 2 for a description of the compiler switches.)

11.2.2 FORTRAN Command Qualifiers

Command qualifiers, each preceded by a slash (/), immediately follow the command name. For example:

```
PDS> FORTRAN/LIST/OBJECT/SWITCHES:(/CK) SOURCE.FTN
```

A programmer specifies command qualifiers in order to modify the function of the FORTRAN command according to the needs of the program. Qualifiers may also be specified merely to affirm default compiler actions. For instance, the example above specifies /OBJECT even though the FORTRAN command produces an object file by default. (See section 11.2.1 for a list of compiler defaults.)

Compiler switches are listed after the /SWITCHES: qualifier. The list of switches must be enclosed in parentheses. The slash preceding each switch separates one from the next within the list. For example:

```
$FORTRAN/SWITCHES:(/CK/CO=7/TR:LINES) PROG1.FTN
```

The possible switches depend on whether the programmer is using FORTRAN IV or FORTRAN IV-PLUS. Both sets of switches are listed in the specification of the FORTRAN command in Part 2.

11.2.3 Examples

The following commands all compile a FORTRAN source file:

1. $FORTRAN/OBJECT/LIST:PRINT RDIN

   Compile the source program RDIN.FTN, create an object file name RDIN.OBJ and create a listing file called PRINT.LST.

2. $FORTRAN/OBJECT/LIST:LPROC1 PROC1

   Compile the source program PROC1.FTN, create an object file named PROC1.OBJ and create a listing file called LPROC1.LST.

3. $FORTRAN/OBJECT/LIST:READ RPRT.FTN

   Compile the source program RPRT.FTN, create an object file named RPRT.OBJ and create a listing file called READ.LST.

Note that the file specifications to the /LIST qualifier need not include an extension. In this case, the system assumes the extension to be LST.

11.3 LINKING OBJECT FILES

The user issues the LINK command to link FORTRAN object files to create an executable task.

11.3.1 The LINK Command

The LINK command invokes the IAS Task Builder to build an executable task from object files generated by the FORTRAN command and from object modules held in user-written and system library files (see Chapter 8, section 8.2).

The IAS Task Builder Reference Manual contains a complete description of the Task Builder.

This section gives information to help the programmer use the LINK command. The user modifies the action of the Task Builder by specifying or defaulting various options.

To link one or more FORTRAN programs using the system default Task Builder switches and options, the user issues the LINK command followed by the list of object files to be linked together into an executable task.

For example:

```
LINK     PERFECT     NUMBER
```

links together the FORTRAN object files PERFECT.OBJ and NUMBER.OBJ.

11.3.1.1 Options - The qualifier /OPTIONS allows the user to specify Task Builder options. In interactive mode the presence of the

qualifier /OPTIONS in the command qualifier list causes the Task Builder to prompt OPTIONS? after the input files have been specified. For example:

```
PDS> LINK/OPTIONS

FILE?  PROG.OBJ,REPORT.OBJ

OPTIONS?
```

The user then enters the options one line at a time. A slash (/) as the first character in a line then terminates the option input and the Task Builder resumes execution.

For example:

```
PDS> LINK/OPTIONS

FILE?  FORT.OBJ, PROG.OBJ

OPTIONS?  ACTFIL=8

OPTIONS?  MAXBUF=160

OPTIONS?  UNITS=9

OPTIONS?  ASG=DT1:7:8:9

OPTIONS?  /
```

In batch mode, the presence of the /OPTIONS qualifier in the command qualifier list causes the Task Builder to expect one or more options to be specified on one or more lines immediately following the command string. A line containing a slash (/) in the first character position terminates the list of options.

For example:

```
$LINK/OPTIONS   PROG.OBJ,    REPORT.OBJ

ACTFIL=8

MAXBUF=160

UNITS=9

ASG=DT1:7:8:9

/
```

The Task Builder options are summarized in a table in the LINK command in Part 2. The table indicates with an 'F' the options that are relevant to FORTRAN programs.

11.3.1.2 Object Modules - The file qualifier /LIBRARY specifies a library file that contains the user-written object modules to be incorporated in the task. The Task Builder automatically searches system object module libraries for referenced modules.

Example:

```
$LINK (FORT.LIB/LIBRARY:(MOD1,MOD2), FORTRAN.OBJ)
```

11.3.1.3 Output Files - The Task Builder does not generate any output files, other than an executable task image, unless the user specifically requests them by supplying the relevant qualifiers. The possible output files and the associated qualifiers are:

Output File	Qualifier
Task image file	/TASK[:filespec]
Memory allocation map file	/MAP[:filespec]
Symbol definition file	/SYMBOLS[:filespec]

11.3.1.4 Example - The following example links three object files.

```
PDS> LINK/TASK:CALC/MAP:CALC/OPTIONS
FILES? RDIN.OBJ,  PROC1.OBJ,  RPRT.OBJ
OPTIONS? UNITS = 5
OPTIONS? ASG=DT2:1:2,TI:3,MT:4:5
OPTIONS? /
PDS>
```

The LINK command links the three object files to create a task image file named CALC.TSK and a map file named CALC.MAP.

11.4 RUNNING THE TASK

A FORTRAN programmer compiles and links a task in separate operations. The RUN command is then used to execute the task image created by the LINK command.

To run a linked FORTRAN task, issue the RUN command and specify the task image file generated by the LINK command.

Examples:

1.
```
PDS> RUN
FILE? CALC
```

2.
```
$RUN CALC
```

Both examples instruct the system to run the task named CALC.TSK.

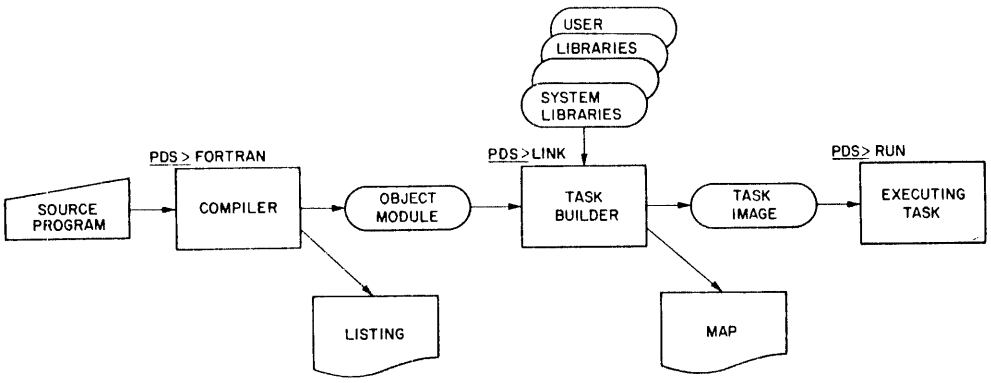

Figure 11-1
Steps in Creating a FORTRAN Program

CHAPTER 12

MACRO-11

A MACRO-11 programmer must complete four steps to transform a MACRO-11 program into an executing task:

1. Create one or more source files,
2. Assemble the source files,
3. Link the assembled. i.e. object, files, and
4. Run the executable task.

This chapter describes how to use IAS commands to perform these steps. It also introduces the On-line Debugging Technique (ODT), a system program which aids in debugging assembled and linked object programs (section 12.5). Consult the IAS/RSX-11 MACRO-11 Reference Manual for information about programming in MACRO-11.

See Chapter 5 for a description of the SUBMIT command. It allows the user to submit a file of IAS commands to a batch processor. A user could create such a file to compile, link and run his task in a single batch job.

12.1 CREATING SOURCE FILES

Either the CREATE command or the EDIT command may be used to create source files. See Chapter 8, section 8.3. The EDIT command has the advantage that it allows the user immediate access to all its editing facilities. To correct errors made while using the CREATE command, however, the user must rely on keyboard facilities or close the file and then issue the EDIT command.

12.2 THE MACRO COMMAND

The MACRO command assembles one or more ASCII source files containing MACRO-11 statements into a single relocatable binary object file. Command qualifiers, including assembler switches, determine the output to be generated by the assembler.

12.2.1 Assembling Source Files

The following command string assembles the source files LOCATE.MAC and FIND.MAC:

1. PDS> MAC

   FILES? LOCATE+FIND

2. $MACRO LOCATE+FIND

Each of the command strings above instructs the system to assemble the source files specified and to produce assembler output as the defaults dictate. Note that the MACRO command requires the source files to be concatenated with a plus sign (+); the assembler does not accept the more common list format, that is, a list enclosed in parentheses, with list items separated by a comma, spaces or tabs. By default, the assembler:

1. Produces an object file which is given the name of the last source file specified and the extension OBJ.

2. In batch mode, produces a listing which is printed on the line printer. Interactive users must specifically request a listing by means of the /LIST qualifier (see section 12.2.2).

12.2.2 Command and File Qualifiers

Command qualifiers, each preceded by a slash (/), immediately follow the command name.

For example:

```
PDS> MACRO/LIST/OBJECT    LOCATE+FIND
```

The programmer specifies file qualifiers immediately after the relevant file specification. For example:

```
$MAC MACLIB.MLB/LIB+TEST
```

the /LIBRARY qualifier instructs the assembler to treat MACLIB.MLB as a macro library file. The /LIST qualifier requests a listing to be sent to the line printer.

A programmer specifies command and file qualifiers in order to modify the function of the MACRO command according to the needs of the program. Qualifiers may also be specified merely to affirm default assembler actions. For instance, the first example above specifies /LIST and /OBJECT even though the MACRO command produces an object file by default. (See section 12.2.1 for a list of assembler defaults.)

The specification of the MACRO command in Part 2 lists all the possible command and file qualifiers. Programmers should consult the IAS/RSX-11 MACRO-11 Reference Manual for a full description.

Example:

```
PDS> MACRO/OBJECT:FINAL

FILE?  ROUT.MAC+MAIN.MAC
```

Assemble the source programs ROUT.MAC and MAIN.MAC to produce an object file named FINAL.OBJ.

12.3 LINKING OBJECT FILES

The user issues the LINK command to link MACRO-11 object files to create an executable task.

See Section 12.5 for information about debugging linked object programs.

12.3.1 The LINK Command

The LINK command invokes the IAS Task Builder to build an executable task from object files generated by the FORTRAN or MACRO command and from object modules held in user-written and system library files (see section 12.3.1.3).

The IAS Task Builder Reference Manual contains a complete description of the Task Builder. This section gives information to help the programmer use the LINK command.

The user may modify the action of the Task Builder by specifying various options. To link one or more MACRO-11 programs with the system default Task Builder switches and options, the user issues the LINK command followed by the list of object files to be linked together into an executable task.

For example:

```
$LINK     REALTIME     ADCONVERT
```

links together the object programs REALTIME.OBJ and ADCONVERT.OBJ.

12.3.1.1 Options - The /OPTIONS qualifier allows the user to specify Task Builder options. In interactive mode the presence of the qualifier /OPTIONS in the command qualifier list causes the Task Builder to prompt OPTIONS? after the input files have been specified. For example:

```
PDS> LINK/OPTIONS

FILE?  PROG.OBJ, REPORT.OBJ

OPTIONS?
```

The user then enters the options one line at a time. A slash (/) as the first character in a line then terminates the list of options and the Task Builder resumes executing.

For example:

```
PDS> LINK/OPTIONS

FILE? MAIN.OBJ,   PROG.OBJ

OPTIONS? TASK=SYSMAN

OPTIONS? UIC=[1,1]

OPTIONS? LIBR=SYSRES:RO

OPTIONS? /
```

In batch mode, the presence of the /OPTIONS qualifier in the command qualifier list causes the Task Builder to expect one or more options to be specified on one or more lines immediately following the command string. The user must specify a single option on each line. A card or line containing a slash (/) in the first character position terminates the list of options.

For example:

```
$LINK/OPTIONS   PROG.OBJ,   REPORT.OBJ

TASK=SYSMAN

UIC=[1,1]

LIBR=SYSRES:RO

/
```

The Task Builder options are summarized in the specification of the LINK command in Part 2. The summary marks the options that are relevant to MACRO programs with the letter M.

12.3.1.2 Object Modules - The file qualifier /LIBRARY specifies the library files that contain the user-written object modules to be incorporated in the task. The Task Builder automatically searches system object module libraries for referenced modules.

Example:

```
$LINK  MACRO.LIB/LIBRARY:(MAC1, MAC2) MACRO.OBJ
```

12.3.1.3 Output Files - The Task Builder does not generate any output files, other than an executable task image, unless the user specifically requests them by supplying the relevant qualifiers. The possible output files and the associated qualifiers are:

Output File	Qualifier
Task image file	/TASK[:filespec]
Memory allocation map file	/MAP[:filespec]
Symbol definition file	/SYMBOLS[:filespec]

12.3.1.4 Example - The following example links three object files to form a task named CALC.TSK.

```
PDS> LINK/TASK:CALC/MAP:CALC/DEBUG/OPTIONS

FILE? (SEG1.OBJ, SEG2.OBJ, MACRO.OBJ)

OPTIONS? UNITS = 5

OPTIONS? ASG=DT2:1:2,TI:3,MT:4:5

OPTIONS? /

PDS>
```

The command string above links the three object files to create a task image file named CALC.TSK and a map file named CALC.MAP. The /DEBUG qualifier instructs the Task Builder to include a debugging aid (i.e. the ODT program, see section 12.5.1) and Task Builder options assign logical unit numbers.

12.4 RUNNING THE TASK

A MACRO-11 programmer assembles and links a task in separate operations. The RUN command is then used to begin execution of the task image created by the LINK command.

When used to execute a MACRO-11 task, the RUN command has no qualifiers and only one parameter, the file specification of the task to be run. The file containing the executable task is the task image file generated by LINK.

Examples:

1.
```
PDS> RUN

FILE? CALC.TSK
```

2.
```
$RUN CALC.TSK
```

Both examples instruct the system to run the task named CALC.TSK.

12.5 DEBUGGING

12.5.1 The On-line Debugging Technique

IAS provides the On-line Debugging Technique (ODT) to help programmers debug linked and assembled object programs. To incorporate ODT in the linked program, the programmer specifies the /DEBUG qualifier to the LINK command (see section 12.3.1.1).

For example:

```
$LINK/DEBUG    MACRO.OBJ
```

The Task Builder then automatically includes ODT in the task image.

The IAS/RSX-11 ODT Reference Manual contains a complete description of ODT. In brief, however, the programmer interacts with ODT and the object program from an interactive terminal to:

1. Print the contents of any location for examination or alteration.
2. Run all or any portion of the object program using the breakpoint feature.
3. Search the object program for specific bit patterns.
4. Search the object program for words which reference a specific word.
5. Calculate a block of words or bytes with a designated value.
6. Fill a block of words or bytes with a designated value.

The breakpoint is one of ODT'S most useful features. When debugging a program, it is often desirable to allow the program to run normally up to a predetermined point, at which time the contents of various registers or locations can be examined and possibly modified. To accomplish this, ODT acts as a monitor to the user program.

During a debugging session you should have the assembly listing of the program to be debugged at the terminal. Minor corrections to the program may be made on-line during the debugging session. The program may then be run under control of ODT to verify any changes made. Major corrections, however, such as a missing subroutine, should be noted on the assembly listing and incorporated in a subsequent updated program assembly.

12.5.2 User-Written Debugging Aids

A programmer may also incorporate a user-written debugging aid in a linked object program. The file containing the debugging aid is specified with the /DEBUG qualifier.

For example:

```
PDS> LINK/DEBUG:DEBUG.OBJ

FILES?  MACRO.OBJ
```

INDEX

PART 2

COMMAND SPECIFICATIONS

COMMAND SPECIFICATIONS

COMMAND FORMAT

The general format of a command is:

[$]command-name[qualifiers][parameteer-1][,...,parameter-n]

The following rules apply:

1. Brackets - In the description of commands in this manual, brackets ([and]) are used to surround optional values. For example:

   COPY[qualifiers]

   indicates that the user does not need to supply any qualifiers to issue a valid COPY command.

2. Dollar Sign ($) - The dollar-sign ($) must appear in position 1 of a command to be executed in batch mode. It may optionally appear in a command executed in interactive mode.

3. Command Names - The command name describes the action the command is to perform. With the exception of LOGIN, LOGOUT, DEASSIGN and DEALLOCATE, which can be abbreviated to 4 letters, all commands can be abbreviated to 3 letters. Additional letters are acceptable, for example, LOGOUT, LOGOU and LOGO are all correct.

4. Parameters - A parameter either describes a value that a command is to use when executing or it further defines the action a command is to take. Interactive users may supply parameters in response to prompts (see Chapter 4, section 4.1). Otherwise, at least one space must separate the first parameter from the command-name; parameters are then separated from each other by one or more spaces and/or a single comma (,).

5. Parentheses and Ellipses - Some commands permit the user to replace a single parameter by a list of values. When this is done the list may be surrounded by parentheses. Parentheses are not required when the parameter being replaced is the only or the last parameter in the command string.

   Examples:

   a. DELETE (A B C)

      The parentheses are optional

   b. APPEND (A B C) D

   The parentheses are required because the parameter being replaced is not the last parameter. (This command specifies that files A, B and C are to be added to the end of file D).

In the description of a command's format, ellipses (three dots "...") indicate that a list of values of the same type may replace a single value.

6. Qualifiers - A qualifier is used to modify the default action of a command. There are defaults for all qualifiers, i.e. they are never required. A qualifier always begins with a slash (/). Both command names and parameters can be qualified.

   Examples:

   ```
   PRINT/DELETE   MYFILE.DAT

   RUN   MYPROGRAM/COBOL
   ```

   Many qualifiers have associated qualifier values. The qualifier is separated from the qualifier value by a colon (:), e.g. KEEP:1. Whenever a qualifier requires a list of values, that list must be enclosed in parentheses, e.g.

   ```
   /BLOCKS:(m-n)
   ```

   A qualifier may not contain any spaces.

7. Continuation Character (-) - A hyphen (-), which may be optionally followed by spaces and/or a comment, is used to indicate that a command is to be continued on the next line.

   Example:

   ```
   COPY A.DAT -

   B.DAT
   ```

8. Comment Character (!) - An exclamation mark delimits the start of a comment. Comments can occur only after the last character of a command or after a hyphen. Comments are for the user's information only and do not affect the processing of the command. Note that a comment character that appears on a continued line must immediately follow the hyphen.

   Examples:

   ```
   COPY A.DAT  B.DAT  !FILE A TO FILE B.

   MOUNT/DENSITY:800 MT: - !  MOUNT MY
   VOLID3  TU10: ! TAPE ON ANY TU10
   ```

9. Concatenation Character (+) - A plus sign (+) indicates concatenation, that is, the records in the file specified on the left of the plus sign are processed followed by the records in the file specified on the right of the plus sign.

   Example:

```
MACRO       A+B
```

   The MACRO-11 statements in file A.MAC followed by the MACRO-11 statements in file B.MAC are read by the MACRO-11 assembler.

DICTIONARY OF PDS COMMANDS

The following PDS commands are specified in this section.

ABORT

ALLOCATE
$ALLOCATE

APPEND
$APPEND

ASSIGN
$ASSIGN

BASIC

COBOL
$COBOL

CONTINUE

COPY
$COPY

CREATE
$CREATE

DEALLOCATE
$DEALLOCATE

DEASSIGN
$DEASSIGN

DELETE
$DELETE

DIRECTORY
$DIRECTORY

DISMOUNT
$DISMOUNT

DUMP
$DUMP

EDIT
$EDIT

$EOD

$EOJ

FORTRAN
$FORTRAN

$JOB

HELP

LIBRARIAN
$LIBRARIAN

LINK
$LINK

LOGIN

LOGOUT

MACRO
$MACRO

MESSAGE
$MESSAGE

MOUNT
$MOUNT

PASSOWRD

PRINT
$PRINT

PROTECT
$PROTECT

QUEUE

RENAME
$RENAME

RUN
$RUN

SET
$SET

SHOW

SUBMIT

TYPE

UNLOCK
$UNLOCK

ABORT

The ABORT command causes the currently suspended user task to be aborted.

FORMAT

PDS> ABORT

The ABORT command has no parameters.

DESCRIPTION

The ABORT command may only be issued after the current user task has been suspended by typing CTRL/C. The ABORT command must then be issued before the system accepts any command other than CONTINUE, HELP, SHOW, MESSAGE or STATUS. Typing ABORT then causes the suspended task to be terminated.

EXAMPLE

```
CTRL/C

TASK SUSPENDED

PDS> ABORT

PDS>
```

ALLOCATE
$ALLOCATE

The ALLOCATE command allocates a specified device to the user and optionally associates a logical name with the device.

FORMAT

PDS> ALLOCATE

RESOURCE? DEVICE

DEVICE? device-name

LOGICAL NAME? logical-name

or

$ALLOCATE DEVICE device-name logical-name

where

device-name is the specification of the device to be allocated to the user.

logical-name is a logical name to be associated with the device.

DESCRIPTION

The user obtains exclusive access to the allocated device.

The system automatically deallocates the device when the user dismounts it or deassigns the last logical unit number to which that device is assigned, unless the user modifies the DISMOUNT or DEASSIGN command with the qualifier /KEEP.

The user may not explicity allocate a system device, that is, a device allocated to all users by the system manager. If device-name does not include a unit number, the system allocates any available device of the specified type and, in interactive mode, prints at the user's terminal the physical unit allocated. In this case, the batch user must define a logical name in order to refer to that device in subsequent commands.

EXAMPLES

1. PDS> ALLOCATE

   RESOURCE? DEVICE

   DEVICE? MT: <ALT>

   LOGICAL NAME? MY0:

   PDS> MOUNT MY0:

   VOLUME-ID? VOL75

   PDS> DISMOUNT/KEEP

   DEVICE? MY0

   VOLUME-ID? VOL74

   PDS> MOUNT MY0: VOL73

   PDS> DISMOUNT

   DEVICE? MY0

   VOLUME-ID? VOL75

2. $ALLOCATE DEVICE MT: LM0

APPEND
$APPEND

The APPEND command adds the contents of one or more input files, in the order in which they are specified, to the end of an output file.

FORMAT

```
PDS> APPEND

FILE? [(]infile-1[,...infile-n)]

TO? outfile
```

or

```
$APPEND [(]infile-1[,...infile-n)]   outfile
```

where

infile — is an input file specification

outfile — is an output file specification

DESCRIPTION

If one or more files in a list of input files is in error, the system ignores the file errors and appends the rest to the output file.

Wild-cards are allowed in any input file specification. All file specifications must include a filename and an extension.

If a version number is not specified, the system assumes the highest version number for the input file, and the highest version plus 1 for the output file.

If one of the specified files is to be input from the user's terminal (TI:), the system transfers all that the user types in after the completed command string. The transfer continues until the user types CTRL/Z to terminate the input file.

EXAMPLES

1.
```
PDS> APPEND (A.OBJ B.OBJ) C.OBJ
```

2.
```
PDS> APPEND
FILE? (ABC.FTN DEF.FTN)
TO? XYZ.FTN
```

3. PDS> APPEND TWO.MAC, ONE.MAC

4. PDS> APPEND (A.EXT B.EXT) D.EXT

5. $APPEND (ABC.DAT,DEF.DAT), XYZ.DAT

ASSIGN
$ASSIGN

The ASSIGN command assigns a device to a logical unit.

FORMAT

PDS> ASSIGN

FILE? device-name

LUN? lun

or

$ASSIGN device-name lun

where

device-name — is the specification of the device to be assigned to the logical unit.

The device must be one allocated to the user by the ALLOCATE or MOUNT command, or one to which all users have access.

lun — is a logical unit number.

DESCRIPTION

Users may assign logical unit numbers at three points:

1. By means of the ASSIGN command before the user runs a task.
2. By means of a Task Builder option when a task is linked (see the IAS Task Builder Reference Manual).
3. From within a program by means of the system directive ALUN$ or OPEN$ or the FORTRAN subroutines ASSIGN and ASNLUN (see the IAS Executive Reference Manual, Volume One).

This command associates a device name with a logical unit number so that user programs can be written independently of specific devices. The assignment applies only to programs executed by means of a batch or interactive RUN command.

If the ASSIGN command associates a device name with a logical unit number, that assignment overrides any made for that logical unit number by the Task Builder. And if an executing program assigns a logical unit, that assignment overrides the action of any ASSIGN command for that logical unit number.

The system automatically deassigns logical units when the associated volume is dismounted or device deallocated. The user may also issue the DEASSIGN command to deassign a device from a logical unit.

EXAMPLES

1. $ASSIGN DPØ: 7

2. PDS> ASSIGN

   FILE? LPØ:

   LUN? 6

3. PDS> ASSIGN DK2:

   LUN? 5

BASIC

The BASIC command invokes the BASIC interpreter.

FORMAT

BASIC

The BASIC command has no parameters.

DESCRIPTION

When the user issues the BASIC command, BASIC indicates that the interpreter is ready to receive a command or program line by typing "READY".

To terminate a BASIC session and return control to PDS, the user types 'BYE' on a new line. The system then prints information about the session and prompts for further PDS commands. For example:

```
BYE

15:57:32 TASK TERMINATION

PDS>
```

Effect of CTRL/C

When the BASIC interpreter is executing a program, CTRL/C causes the system to stop execution after the current line. The terminal displays the number of the last line executed and the user may then issue further BASIC commands.

CTRL/C typed during the execution of a BASIC LIST or SAVE command or an immediate mode statement stops the execution of those commands or statements. It has no effect on the execution of other BASIC commands.

COBOL
$COBOL

The COBOL command compiles a COBOL source program.

FORMAT

PDS> COBOL[qualifiers]

FILE? filespec

or

$COBOL[qualifiers] filespec

where

filespec is a specification of the file containing the COBOL program. The specification must contain a filename. If the extension is omitted, the system assumes it to be CBL.

qualifiers are one or more of the following:

Qualifier	Meaning
/OBJECT[:filespec]	Produce an object file, named according to filespec if it is supplied (no wild-cards allowed). The default extension is OBJ. /OBJECT is the default qualifier.
/NOOBJECT	Do not produce an object file.
/LIST[:filespec]	Produce a listing file named according to filespec if it is supplied (no wild-cards allowed). The default extension is .LST. /LIST is the default condition for batch mode.
/NOLIST	Do not produce a listing file (the default condition for interactive mode).
/SWITCHES:(switches)	Apply the specified COBOL compiler switches. See the section called Compiler Switches below.

Defaults

Object File - If the qualifier /OBJECT is specified without a file specification, the object file is given the name of the source file and the extension .OBJ. The system default is /OBJECT.

Listing File - If /LIST is supplied without a filespec in interactive mode, the listing file is sent to the line printer. In batch mode, the system assumes the /LIST qualifier by default and automatically prints the listing on the line printer.

Compiler Switches

The COBOL command includes compiler switches that permit the user to tailor the compilation listing to meet particular needs. Other switches are available at execution time that may be interrogated by the object program.

A list of switches and their meanings follow:

Switch	Meaning	Default
/ERR:n	Suppress the printing of diagnostics with a severity number of less than n. The range of n must be Ø<n<2. The switch cannot suppress severity 2 (fatal) diagnostics. (An entry of 2 suppresses the printing of all severity numbers that are less than 2.)	/ERR:Ø
/ACC:n	Produce an object program only if the source program contains diagnostics with severities equal to or less than n. The range of n must be Ø<n<2.	/ACC:1
/MAP	Produce a Data Division map showing the memory addresses for Data Division entries.	/NOMAP
/LOD	Print the starting and ending addresses of the Data and Procedure Division blocks. Indicate the number of PD blocks available and the version number of the compiler that produced this object program.	/NOLOD
/CVF	The source program is in conventional format (i.e., 8Ø-character images with Area A beginning in character position 8).	
/USW:n...:m	Turn on switches n through m. The system sets to OFF all of the switches that are not named. The switches n through m must be in the range 1 to 16 (decimal) or 1 to 2Ø (octal). Switch numbers that are expressed in decimal must be followed by a decimal point (for example, switch 14 (decimal) should be expressed as /USW:14.). If the decimal point is omitted, the system interprets /USW:14 (decimal) as an octal number and sets switch 12 on. Each switch in a series must be separated from the other switches by colons.	

Examples

1. PDS> COBOL COBPROG.CBL

2. PDS> COBOL/SWITCHES:(/MAP/LOD)

   FILE? COBPROG.CBL

3. $COBOL BATCHCOB

CONTINUE

The CONTINUE command causes the currently suspended user task to resume execution.

FORMAT

PDS> CONTINUE

The CONTINUE command has no parameters.

DESCRIPTION

The CONTINUE command may only be issued after the user task has been suspended by typing CTRL/C. Typing CONTINUE reactivates the currently suspended task.

EXAMPLE

```
CTRL/C

TASK SUSPENDED

PDS> CONTINUE
```

COPY
$COPY

The COPY command copies:

1. one file to another file,
2. a group of files to another group of files,
3. the concatenation of a number of files to a single file.

FORMAT

PDS> COPY[qualifiers]

FROM? infile[file-qualifier]

TO? outfile[file-qualifier]

or

$COPY[qualifiers] infile[file-qualifier], outfile[file-qualifier]

where

infile — is an input file specification. Concatenated files are linked by a plus sign (+).

For example:

filespec+filespec+filespec+....

outfile — is an output file specification.

qualifiers — are one or more of the following:

Qualifier	Meaning
/ALLOCATION:n	Allocate n blocks to the output file
/CONTIGUOUS	Make the output file contiguous
/OWN	Copied file(s) owned by output file (UFD).
/REPLACE	Replace the existing output file, if any.

file-qualifier modifies the specification of a foreign file in DIGITAL'S DOS or RT-11 format. The qualifiers are:

/DOS
/RT11

DESCRIPTION

If infile or outfile has a filename then it must also have an extension.

If the version number is omitted from the input file then the highest version number is used. If it is omitted from the output file then the highest version number plus 1 is used.

Wild-cards are allowed whenever an input file specification does not describe concatenated files. If any of the filename, extension or version fields of the output file contain a wild-card, all fields must be wild; the version field, however, may be omitted. If one part of the output file UFD is a wild-card, both parts must be wild.

If /DOS or /RT11 modifies either file specification, then the input files may not be concatenated and the output filename and extension must be wild (that is, foreign files may not be renamed).

If the user enters infile from the user's terminal (TI:), the system transfers to the output file all that the user types in after the completed command string. The transfer continues until the user types CTRL/Z to terminate the input file.

If either infile or outfile is not in Files-11 format, its specification must be modified by either /DOS or /RT11. The system does not accept any other foreign formats.

EXAMPLES

1. PDS> COPY A.CBL B.CBL

2. $COPY E.EXT, F.EXT

3. PDS> COPY
   FROM? E.EXT
   TO? F.EXT

4. PDS> COPY/OWN DKØ:[*,*]*.*
   TO? DK1:[*,*]*.*

5. PDS> COPY DATA.DAT DTØ:*.*

CREATE
$CREATE

The CREATE command creates a file and copies into it source lines following the command in a batch stream or input entered from a terminal.

FORMAT

```
PDS> CREATE

FILE? filespec[/PROTECTION:(code)]

terminal-input

CTRL/Z
```

or

```
$CREATE[/DOLLARS] filespec[/PROTECTION:(code)]
source-file-lines
```

where

filespec	is a single file specification. Wild-cards are not allowed. A file extension must be included.
terminal-input	is everything that the user types at the terminal between the command terminator and CTRL/Z (see Description below).
source-file-lines	are the source-lines to be copied into the new file.
/PROTECTION:(code)	assigns protection as specified in code to the newly created file (see Chapter 6, section 6.1.3).
/DOLLARS	(Valid only in batch mode) specifies that the file will be terminated by $EOD.

DESCRIPTION

Batch Mode

In batch mode the text to be placed in the new file follows the command. Any $ command terminates the file unless the CREATE command string includes the qualifier /DOLLARS, which specifies that only the command $EOD can terminate the file.

Interactive Mode

The CREATE command reads the input to the new file from the user's terminal. Pressing CTRL/Z terminates the file.

The CREATE command has the same function as a COPY command that specifies TI: as the device in the input file specification.

EXAMPLES

1. PDS> CREATE

   FILE? MYDATA.DAT;5

   ABCD

   EFGH

   CTRL/Z

   PDS> CREATE ANOTHER.DAT/PROTECTION:(OW:RW)

   CTRL/Z

   PDS>

2. $CREATE/DOLLARS DEBUG.MAC

   .
   .
   .

   $EOD

DEALLOCATE
$DEALLOCATE

The DEALLOCATE command deallocates a specified device.

FORMAT

PDS> DEALLOCATE

RESOURCE? DEVICE

DEVICE? device-name

or

$DEALLOCATE DEVICE device-name

where

device-name is the device specification or the logical name of the device to be deallocated.

DESCRIPTION

Normally the system automatically deallocates a device when the user dismounts the volume on it or deassigns it from a logical unit number. However, when the user has issued the ALLOCATE command to obtain access to a non-mountable device that has not been assigned to a logical unit, the DEALLOCATE command must be used to release it. It may also be used after a DEASSIGN/KEEP or DISMOUNT/KEEP command.

EXAMPLES

1. PDS> DEALLOCATE DEVICE LPØ:
2. $DEALLOCATE DEVICE DD:

DEASSIGN
$DEASSIGN

The DEASSIGN command dissociates a device from a logical unit.

FORMAT

PDS> DEASSIGN[/KEEP]

LUN? lun

or

$DEASSIGN[/KEEP] lun

where

/KEEP inhibits any deallocation or dismounting of the associated device.

lun is the logical unit number to be deassigned.

DESCRIPTION

If the specified logical unit number is the last to which a device is assigned, the device is dismounted or deallocated unless the user specifies the command qualifier /KEEP.

EXAMPLES

1. PDS> DEASSIGN
   LUN? 7

2. $DEASSIGN/KEEP 3

DELETE
$DELETE

The DELETE command deletes one or more specified files.

FORMAT

```
PDS> DELETE[/KEEP:n]

FILE?  filespec-1[file-qualifier][,...filespec-n]
```

or

```
$DELETE[/KEEP:n] filespec-1[file-qualifier][,...filespec-n]
```

where

KEEP[:n]	prevents the latest n versions of a specified file from being deleted. It can only be used when the version field in a file specification is omitted or wild. If n is omitted, it is assumed to be 1.
filespec	is the file specification of a file to be deleted. A filename and an extension must be specified. Wild-cards are allowed in the UFD, filename, extension and version fields. The version field must be supplied.
file-qualifier	modifies the specification of a foreign file in DIGITAL'S DOS or RT-11 format. The qualifiers are: /DOS /RT11

DESCRIPTION

The user cannot recover a deleted file.

In order to delete a file in DOS or RT11 format, the user must modify the file specification with the /DOS or /RT11 file qualifier.

EXAMPLES

1.
```
PDS> DELETE, (A.EXT;1, B.EXT;1, DK0:C.*;*)
```

2.
```
PDS> DELETE/KEEP:1
FILE? DK1:[200,200]*.XYZ;*
```

3. $DELETE/KEEP DK0:[200,200]*.LIS;*

4. PDS> DELETE DT0:TEST.MAC/DOS

DIRECTORY
$DIRECTORY

The DIRECTORY command lists details about a file or a group of files at a specified output device or to a specified file. Command qualifiers allow the user to request greater or less detail.

FORMAT

PDS> DIRECTORY[qualifier(s)]

FILE? filespec-1[file-qualifier][,..filespec-n]]

or

$DIRECTORY filespec-1[file-qualifier][,...filespec-n]

where

filespec — is a file specification that indicates the directory entries to be listed. Wild-cards are allowed.

If no files are specified, the system lists information about all the files in the user's directory.

qualifier(s) — are one or more of the following:

Qualifier	Meaning
/OUTPUT:outfile	List information in the specified output file.
/BRIEF	List only the name, type and version of the file(s).
/FREE	Show free space available within the user's directory.
/FULL	List all the following file details:

1. Name, extension and version
2. File identification number in the format:(file number, file sequence number)
3. Number of blocks used or allocated.
4. File code

   null = non-contiguous
   C = contiguous
   L = locked

5. Creation time and date

6. Owner UIC and file protection in the format: (group, owner) (system, owner, group, world)

7. Date and time of the last update and the number of revisions.

file-qualifier modifies a foreign file in DIGITAL'S DOS or RT-11 format. The qualifiers are:

/DOS
/RT11

DESCRIPTION

By default, the DIRECTORY command lists at the user's terminal (interactive mode) or in the user's output stream (batch mode) the name, extension, version, size, file code, and date and time of creation of all the files in the user's directory.

The user may not examine the files in a directory for which he does not have read access.

To interrogate the directories of DOS or RT-11 volumes, the user must modify the file specification with the /DOS or /RT-11 file qualifier.

The directory listing of a DOS or RT-11 file corresponds to the directory format of the foreign volume. The qualifiers /BRIEF and /FULL are not valid when requesting foreign directory information.

EXAMPLES

1. PDS> DIRECTORY <ALT>

   FILE? MATRIX.DAT/DOS

2. PDS> DIR/FULL/OUTPUT:LP0: <ALT>

   FILE? DK1:[200,200]*.LST

3. $DIR/BR FRED.*

4. $DIRECTORY DK1:*.*/RT11

DISMOUNT
$DISMOUNT

THE DISMOUNT command causes the volume on the specified device to be dismounted.

FORMAT

PDS> DISMOUNT[/KEEP]

DEVICE? device-name

VOLUME-ID? volume-label

or

$DISMOUNT[/KEEP] device-name volume-label

where

/KEEP	instructs the system not to deallocate the device
device-name	is the physical or logical name of the device to be dismounted.
volume-label	is an optional parameter that specifies the label of the volume to be dismounted (see the MOUNT command).

DESCRIPTION

If the user does not specify /KEEP, the system dismounts the volume on the device, deallocates the device, and deassigns it from any logical unit number.

The command qualifier /KEEP prevents the system from deallocating the device.

EXAMPLE

1. PDS> DISMOUNT

   DEVICE? MY0:

   VOLUME-ID?

2. $DISMOUNT/KEEP TU1: ACCTS

DUMP
$DUMP

The DUMP command produces a printed listing of the contents of a file that ignores any print formatting characters that may appear in the records. The listing is printed at the user's terminal by default, but the user may specify a different output device.

FORMAT

PDS> DUMP[qualifier(s)]

FILE? filespec

or

$DUMP[qualifier(s)] filespec

where

filespec — is the specification of the file or device to be dumped.

qualifier(s) — are one or more of the following command qualifiers:

Qualifier	Meaning
/OUTPUT:filespec	Output the listing to the specified file or device.
/ASCII	The /ASCII switch specifies that the data should be listed in ASCII mode. The control characters (0 - 37) are printed as ^ followed by the alphabetic character corresponding to the character code +100(octal). For example, bell (code 7) is printed as ^G (code 107). Lower case characters (140 - 177) are printed as % followed by the corresponding upper case character (character code -40)
/BLOCKS:(m-n)	Specifies the first (m) through the last (n) logical or virtual blocks to be listed, where m and n are octal numbers. If either m or n is greater than 16 bits (that is, greater than 177777) then the user must specify it as two numbers as follows: (a,b) where a is the first 16 bits and b is the second 16 bits. If the /BLOCKS:(m-n) switch is specified as /BLOCK:0 in file mode, no physical blocks will be listed.

This is useful when the user wishes tp list only the header portion of the file. (See the /HEADER switch below).

This qualifier is necessary in device mode; it specifies the range of physical blocks to be listed.

/BYTE

The /BYTE qualifier specifies that the data should be listed in byte octal format.

/HEADER

If specified, /HEADER causes the file header as well as the specified portion of the file to be listed.

If just the header portion of the file is required, the user can specify /HEADER/BLOCKS:0.

/START

This qualifier gives the user only the starting block number of the file and an indication of whether or not it is contiguous.

Example:

```
DUMP/START DK0:RICKSFILE.DAT;3
STARTING BLOCK NUMBER = 0.135163 C
```

File RICKSFILE.DAT, version 3 is a contiguous file starting at block no. 0,135163,
(See /BLOCKS:(m-n) for a description of block and number format.)

/NUMBER[:n]

This qualifier allows control of line numbers. Line numbers are normally reset to zero whenever a block boundary is crossed. The /NUMBER[:n] qualifier allows lines to be numbered sequentially for the full extent of the file; i.e., the line numbers are not reset when block boundaries are crossed. The optional value (:n) allows the user to specify the value of the first line number. The default is 0.

DESCRIPTION

The DUMP command operates in either one of two modes:

1. File Mode

   In file mode, the user specifies a file; all, or a specified range (see(/BLOCKS:(m-n)) of virtual blocks of the named file are listed. Virtual blocks refer to the physical blocks of data in the files. The blocks are numbered from 1 through n, where the first block is 1 and the last block in the file is numbered n. The input volume must be mounted and it must contain named files.

2. Device Mode

   In device mode, the user specifies a device; then a specified range (/BLOCKS:(m-n)) of physical blocks to be listed.

   a. The /BLOCKS:(m-n) qualifier is required.

   b. Physical blocks refer to the actual 512-byte blocks on disk and DECtape, and physical records on magtape and physical records on magtape and cassette. The DUMP command handles physical records up to 2048 bytes in length.

   c. Physical blocks are numbered from 0 to n, where n is the last logical block on the device.

   d. The volume to be listed must be mounted as FOREIGN.

EXAMPLES

1. PDS> DUMP MYFILE.DAT

2. PDS> DUMP/ASCII

   FILE? MYDATA.DAT

3. $DUMP A.MAC;4

EDIT
$EDIT

The EDIT command invokes one of the following IAS text editors:

1. The Line Text Editor (EDI), an editor primarily for interactive use

2. The Source Language Input Program and Editor (SLIPER), a batch-oriented editor.

Chapter 7 describes how to use both editors. The IAS Editing Reference Manual specifies both in full.

FORMAT

PDS> EDIT[/editor][qualifier(s)]

FILE? filespec

or

$EDIT[/editor][qualifier(s)] filespec

where

/editor — is either:

/EDI which invokes the Line Text Editor, or

/SLIPER which invokes the batch editor SLIPER

The default is /EDI

qualifier(s) — are one or more command qualifiers that are only valid if /SLIPER has been specified. The qualifiers are described in detail in Chapter 7, Section 7.2.1. They are:

Qualifier	Default
/OUTPUT[:filespec]	/OUTPUT
/NOOUTPUT	
/LIST[:filespec]	/LIST (batch mode)
/NOLIST	/NOLIST (interactive mode)
/AUDIT	/AUDIT
/NOAUDIT	
/BLANK	/BLANK
/NOBLANK	
/DOUBLE	/NODOUBLE
/NODOUBLE	

filespec — is the specification of an existing file to be edited or a new file to be created. An extension must be included.

The Line Text Editor (EDI)

The Line Text Editor is described in Chapter 7. A complete specification is contained in the IAS Editing Utilities Reference Manual. This section lists all the editor commands that can be issued once the user has invoked the Line Text Editor.

ADD A[DD] string	Add the text specified by "string" to the end of the current line.
ADD AND PRINT AP string	Same as ADD, except the new current line is printed out.
BEGIN B[EGIN]	Sets the current line pointer to the top of the block buffer or input file.
BLOCK ON or OFF BL[OCK][ON] or [OFF]	Switch editing modes.
BOTTOM BO[TTOM]	Sets the current line pointer to the bottom of block buffer or input file.
CHANGE [n]C[HANGE] /string-1/string-2	Search for string-1 and replace it with the text specified in string-2. n allows the user to repeat the command, thus allowing string-2 to be substituted for string-1 n times within the current line.
CLOSE CL[OSE]	Transfer the remaining lines in the block buffer and the input file into the output file, then close both the input file and the output file.
CLOSES CLOSES	Close secondary input file, and begin selecting lines from the input file.
CLOSE AND DELETE CDL	Same as the CLOSE command except that

	the input file is deleted.
CONCATENATION CHARACTER CC character	Change command concatenation character to the specified character (default is &).
DELETE D[ELETE] [n] or [-n]	Delete the current and next n-1 lines, if n is positive; delete n lines preceding the current line, but not the current line, if n is negative.
DELETE AND PRINT DP [n] or [-n]	Same as DELETE except that the new current line is printed out.
END E[ND]	Same as the BOTTOM command.
ERASE ERASE [n]	Erase the entire block buffer, the current line, and the next n blocks.
EXIT EX[IT]	Same as CLOSE command.
EXIT AND DELETE EDX	Exit from the editing session, close the output file, delete the input file.
FORM FEED FF	Insert form feed into block buffer.
FILE FI[LE] filespec	Transfer lines from the input file to the file specified by filespec.
FIND [n]F[IND] string	Find the line starting with "string" or, if n is specified the nth line starting with "string".

INSERT I[NSERT] [string]	Insert "string" immediately following the current line. If "string" is null, EDI enters Input Mode.
KILL KILL	Terminate this editing session; close input and output files; delete the output file.
LINE CHANGE [n]LC /string-1/string-2	Same as CHANGE except that all occurrences of string-1 in the current line are changed to string-2.
LIST ON TERMINAL LI[ST]	Print on user terminal all lines in block buffer or all remaining lines in input file, starting with current line.
LOCATE [n]L[OCATE] string	Search the block buffer for "string" or, if n is specified the nth occurrence of "string".
MACRO MA[CRO] x definition	Define macro x to be "definition".
MACRO CALL MC[ALL]	Retrieve macros from the latest version of file MCALL.EML.
MACRO EXECUTE [n]Mx[a]	Execute Macro x for n executions passing it the numeric argument a.
MACRO IMMEDIATE [n]<definition>	Immediate Macro - this allows the user to define and execute a macro in one step.
NEXT N[EXT] [n] or [-n]	Establish a new current line + or - n

	lines from the current line.
NEXT PRINT NP [n] or [-n]	Next Print; same as Next command, but the new current line is printed out.
OLD PAGE OL[DPAGE] n	Back up to page n.
OPENS OPENS filespec	Open secondary input file.
OUTPUT ON or OFF OU[TPUT] [ON] or [OFF]	Turn output on or off.
OVERLAY O[VERLAY] [n]	Delete the current line and the next n-1 lines, and enter Input Mode.
PAGE PAG[E] [n]	Enter block edit mode, if not already in block edit mode, and read page n into the block buffer.
PAGE FIND [n]PF[IND] string	Identical to FIND command except that it searches successive pages until the nth occurrence of "string" is found.
PAGE LOCATE [n]PL[OCATE]string	Same as LOCATE command, except that successive pages are searched for the value specified by "string".
PASTE PA[STE] /string-1/string-2	The same as the LINE CHANGE command except that all lines in the remainder of the input file or block buffer are searched for string-1. Wherever found, string-1 is replaced with string-2.

PRINT P[RINT] [n]	Print out the next line, and the next n-1 lines, on the terminal.
READ REA[D] [n]	Read the next n pages into the block buffer.
RENEW REN[EW] [n]	Write the current buffer, and read in the next page.
RETYPE R[ETYPE] [string]	Replace the current line with the text of "string". If "string" is null; the line is deleted.
SAVE SA[VE] [n] [filespec]	Save the current line, and the next n-1 lines, in the file specified by filespec.
SEARCH & CHANGE SC /string-1/string-2	Search for string-1, in the block buffer or input file starting with the line following the current line. When string-1 is found, replace all occurrences in line with string-2.
SELECT PRIMARY SP	Select primary input file.
SELECT SECONDARY SS	Select secondary input file.
SIZE SIZE n	Specify maximum number of lines to be read into the block buffer on a single READ.
TAB ON or OFF TA[B] [ON] or [OFF]	Turn automatic tabbing on or off.

TOP
T[OP]

Same as BEGIN command.

TOP OF FILE
TOF

Returns to the top of the input file, in block edit mode, and saves all pages previously edited.

TYPE
TY[PE] [n]

Same as PRINT command except that the current line pointer does not change.

UNSAVE
UNS[AVE] [filespec]

Retrieve the lines which were previously saved on filespec and insert them immediately following the current line.

VERIFY ON or OFF
V[ERIFY] [ON] or [OFF]

Allows user to select whether or not locative and change commands are to be verified.

WRITE
W[RITE]

Write the current block to the output file, and erase the contents of the buffer.

The Source Language Input Program and Editor (SLIPER)

The SLIPER edit control characters are as follows:

Character	Function
-(minus)	Indicates that an editing function is to be performed, with reference to the line number(s) specified. -n Insert text following line n. -n,n Delete line n. -n,m Delete lines n through m inclusively.
/(slash)	The slash is placed in the first position of a line to indicate that the editing of a file is completed.
@(at)	The @ character is put in the first location of a line to indicate that SLIPER is to seek input from an indirect file. The user must specify the indirect file immediately after the @ sign; for example: @DK2:DKSFIL.CMD instructs SLIPER to read input from the file DKSFIL.CMD on physical device unit DK2:. Indirect files are more fully described in Section 7.2.4.
<(less than)	The < character is used when entering a line that begins with one of the special edit control characters. It causes the line to be shifted one character to the left, with the result that < is deleted, and the desired control character on the line.

$EOD

The $EOD (End of Data) command terminates a data stream or the input to a file created by a $CREATE/DOLLARS command.

FORMAT

```
$EOD
```

The command has no parameters.

EXAMPLE

```
$CREATE/DOLLARS BATCH.CMD
$JOB WILSON TESTRUN 2
$MOUNT DK: VOL273 DDØ:
$ASSIGN DDØ: 3
$RUN TEST
$DISMOUNT DDØ:
$EOJ
$EOD
```

This example uses $EOD to terminate a file of batch commands (an indirect file). The /DOLLARS qualifier instructs the system to accept the following lines of text as input to the file rather than batch commands to be processed.

$EOJ

The $EOJ (End of Job) command terminates a batch job.

FORMAT

```
$EOJ
```

The command has no parameters.

DESCRIPTION

THE $EOJ command must be the last command in a batch job command stream.

EXAMPLE

```
$JOB WILSON TESTRUN 2
$MOUNT DK:  TEST DDØ:
$ASSIGN DDØ: 7
$RUN TEST
$DISMOUNT DDØ:
$EOJ
```

FORTRAN
$FORTRAN

The FORTRAN command invokes a FORTRAN compiler to compile one FORTRAN-IV or FORTRAN-IV PLUS source file. Command qualifiers control output file options and subsequent processing.

FORMAT

PDS> FORTRAN[qualifier(s)]

FILE? filespec

or

$FORTRAN[qualifier(s)] filespec

where

filespec — is the specification of a source program file to be compiled.

If the extension is omitted, the system assumes it to be FTN. No wild-cards allowed.

qualifier(s) — are one or more of the following command qualifiers:

Qualifier	Meaning
/FOR	Invoke the FORTRAN-IV compiler. Applicable to systems that have both FORTRAN IV and FORTRAN IV PLUS compilers. If omitted, the system invokes its default compiler.
/F4P	Invoke the FORTRAN IV-PLUS compiler. Applicable to systems that have both FORTRAN IV and FORTRAN IV PLUS compilers. If omitted, the system invokes its default compiler.
/LIST[:filespec]	Produce a listing file; re-name as indicated. If an extension is omitted from filespec, the system assumes it to be .LST.
/NOLIST	Do not produce a listing file.
/OBJECT[:filespec]	Produce an object file; re-name as specified.
/NOOBJECT	Do not produce an object file.
/SWITCHES:(/SW1.../SWn)	Use specified FORTRAN IV or FORTRAN IV-PLUS switch options. For further details, see. the appropriate FORTRAN User's Guide.

DEFAULTS

1. By default, the compiler produces an object file with the name of the source file and the extension OBJ.

2. By default, the compiler:

   a. Sends a listing to the line printer in batch mode.

   b. Does not produce a listing file in interactive mode. If /LIST is specified without a filespec then the listing file is printed on the line printer.

FORTRAN-IV Switches

Switch	Default	Description
/LI:n	/LI:3	Specifies the listing options. The argument n is encoded as follows: /LI:0 or /NOLI — list diagnostics only /LI:1 or /LI:SRC — list source program and diagnostics only /LI:2 or /LI:MAP — list storage map and diagnostics only /LI:4 or /LI:COD — list generated code and diagnostics only Any combination of the above list options may be specified by summing the numeric argument values for the desired list options. For example: /LI:7 or /LI:ALL requests a source listing, a storage map listing, and a generated code listing. If this switch is omitted the default list option is /LI:3, source and storage map.
/DE	/NODE	Compile lines are with a D in column one. These lines treated as comment lines by default.
/EX	/NOEX	Read a full 80 columns of each record in the source file. Only the first 72 columns are read by default
/ID	/NOID	Print FORTRAN identification and version number. The default (/NOID) causes the identification and version number not to be printed.

Switch	Default	Description
/OP	/OP	Enable the Common Subexpression Optimizer (CSE). In general the CSE optimizer will make the program run faster. However, the size of the program may be different than with no optimization.
/SN	/SN	Include internal sequence numbers (ISN). The option reduces storage requirements for generated code and slightly increases execution speed but disables line number information during Traceback.
/I4	/NOI4	Two word default allocation for integer variables. Normally, single storage words will be the default allocation for integer variables not given an explicit length specification (i.e., Integer*2 or integer*4). Only one word is used for computation.
/VA	/VA	Enable vectoring of arrays (see section 2.5 of the FORTRAN IV User's Guide).
/WR	/WR	Enable compiler warning diagnostics.

Switch default summary:

(/LI:3/NODE/NOEX/NOID/OP/SN/NOI4/VA/WR)

FORTRAN-IV Plus Switches

Switch	Default	Description
/CK	/NOCK	Code is generated to check that all array references are within the array bounds specified by the program. Individual subscripts are not checked against dimensional specifications.
/CO:n	/CO:5	A maximum of n continuation lines is permitted in the program, Ø<n<99. The default value is n=5. Note that n may be expressed either in octal or decimal radix. If a decimal point follows the number, it is interpreted in decimal radix; otherwise, it is interpreted in octal radix.
/DE	/NODE	Compile lines with a D in column one. These lines are treated as comment lines by the default /NODE (see the FORTRAN Language Manual).
/ID	/NOID	Print FORTRAN IV-PLUS identification and version number.

Switch	Default	Description
/I4	/NOI4	Allocates two words for default length of Integer and Logical variables. Normally, single storage words will be the default allocation for all Integer or Logical variables not given an explicit length definition (i.e., INTEGER*2, LOGICAL*4). See Section 3.3 of the FORTRAN IV-PLUS User's Guide.
/LI:n	/LI:1	Specifies listing options; Ø<n<3. The argument is code as follows: n=Ø minimal listing file: diagnostic messages and program section summary only n=1 (default) source listing and program section summary n=2 source listing, program section summary and symbol table n=3 source listing, assembly code, program section summary, and symbol table
/TR:XXX	/TR:BLOCKS	The /TR switch controls the amount of extra code included in the compiled output for use by the OTS during error traceback. This code is used in producing diagnostic information and in identifying which statement in the FORTRAN source program caused an error condition to be detected at execution. /TR:XXX can have the following forms: /TR Same as TR:ALL /TR:ALL Error traceback information is compiled for all source statements, and function and subroutine entries. /TR:LINES Same as ALL option. /TR:BLOCKS Traceback information is compiled for subroutine and function entries and for selected source statments. The source statements selected by the compiler are initial statements in sequences commonly called basic blocks. The compiler treats such a sequence of statements as a unit for performing certain types of optimization. Basic blocks generally begin at each labelled statement, each DO statement, and so on. /TR:NAMES Traceback information is compiled only for subroutine and function entries.

Switch	Default	Description
	/TR:NONE	No traceback information is produced.
	/NOTR	Same as NONE.

The switch setting /TR is generally advisable during program development and testing. The default setting /TR:BLOCKS is generally advisable for most programs in regular use. The setting /NOTR may be used for obtaining fast execution and smallest code, but it provides no information to the OTS for diagnostic message traceback.

Compiler switch default summary:

(/NOCK/CO:5/NODE/ID/NOI4/LI:1/TR:BLOCKS)

FURTHER INFORMATION

For further information on the use of the FORTRAN sytems, refer to the following documents:

PDP-11 FORTRAN Language Reference Manual

IAS/RSX-11 FORTRAN-IV User's Guide

FORTRAN IV-Plus User's Guide

EXAMPLES

1. PDS> FORTRAN NEWFILE
2. PDS> FORTRAN/SW:(/CK/CO:7) FILES.FTN
3. $FORTRAN/OBJ:YRFILE.OBJ MYFILE

$JOB

The $JOB command initiates a batch job.

FORMAT

```
$JOB username jobid time-limit
```

where

username	is an alphanumeric string 1 to 12 characters long which is unique to the user.
jobid	is an alphanumeric string 1 to 12 characters long which identifies the job. The system prints the jobid at the beginning and end of the job's printed output.
time-limit	is the time-limit (in integer format) in minutes of the job's elapse time.

DESCRIPTION

THE $JOB command must be the first command in a batch job command stream.

EXAMPLES

1. $JOB PIERCE JOBONE 2

2. $JOB PARKER ANALYSIS 3

HELP

The HELP command displays information at an interactive terminal to assist the user in issuing PDS commands.

FORMAT

PDS> HELP

The command has no parameters

DESCRIPTION

The precise information displayed depends on the users current activity.

LIBRARIAN
$LIBRARIAN

The LIBRARIAN command allows the user to create, delete and maintain object module libraries and MACRO-11 macro libraries.

FORMAT

```
PDS> LIBRARIAN
OPERATION?  operation[qualifiers]
LIBRARY?  libspec
[librarian prompt?  text]
```

or

```
$LIBRARIAN operation[qualifiers] filespec [text]
```

where

operation	is the librarian operation to be performed. The operations are: COMPRESS CREATE DELETE INSERT LIST REPLACE
libspec	is a file specification of the library file on which the operation is to be performed.
qualifiers librarian prompt text	are all dependent on the operation specified and are described accordingly below

Library Types

There are two types of library:

- those containing object modules (object module libraries) and
- those containing macros (macro libraries).

Object module libraries are created with a default extension of .OLB. Each object module inserted into the library has its module name (taken from the .TITLE statement) added to the module name table (MNT) and its entry points (globals) added to the entry point table (EPT).

Macro libraries are created with a default extension of .MLB. Each macro inserted into the library has its module name (taken from the .MACRO statement) added to the module name table (MNT).

Restrictions

The following restrictions apply to the handling of object modules:

1. The size of a module is limited to 65,536 words.
2. The size of the library file is limited to 65,536 words.
3. Tables and contiguous space should be allocated the maximum anticipated size. Expanding space allocations require the COMPRESS operation to copy the entire file.
4. A fatal error results if an attempt is made to insert a module into a library which contains a differently named module with the same entry point.

COMPRESS

The COMPRESS operation physically deletes logically deleted (by the DELETE operation) modules in the file specified and re-arranges the file, putting all free space at the end of the file, where it is available for new module inserts.

Format

```
PDS> LIBRARIAN
OPERATION? COMPRESS[qualifiers]
LIBRARY? libspec
NEW LIBRARY? newlibspec
```

or

```
$LIBRARIAN COMPRESS[qualifiers] libspec newlibspec
```

where

libspec is a specification of the library file to be compressed (no wild-cards allowed).

newlibspec is a specification of the compressed library file (no wild-cards allowed).

The operation qualifiers are as follows:

Qualifier	Description	Default
/SIZE:n	The size in 256-word blocks of the compressed file.	100
/EPT:n	The number of entries to allocate in the entry point table (not greater than 1024). A macro library has no entry point table. n is rounded up to the nearest multiple of 64.	512 (object) 0 (macro)

/MNT:n	The number of entries to allocate in the module name table (not greater than 1024). n is rounded up to the nearest multiple of 64.	256

Examples

1. PDS> LIBRARIAN COMPRESS/SIZE:150

   LIBRARY? PEEK.OLB

   NEW LIBRARY? PEEK2.OLB

   The object library file PEEK.OLB is compressed to 150 blocks with 512 EPT entries and 256 MNT entries by default. The compressed file is called PEEK2.OLB.

2. $LIBRARIAN COMPRESS FREAN.MLB FREAN2.MLB

   The macro library file FREAN.MLB is compressed to 100 blocks with no EPT entries and 256 MNT entries by default. The compressed file is called FREAN2.MLB

CREATE

The CREATE operation allocates a contiguous library file on a direct access device (e.g. disk), and initializes the library header and tables.

Format

```
PDS? LIBRARIAN
OPERATION? CREATE/[qualifiers]
LIBRARY? libspec
FILE? infile-1[,...infile-n]
```

or

```
$LIBRARIAN CREATE[qualifiers] libspec infile-1[,...infile-n]
```

where

libspec — is a specification of the library file to be created (no wild-cards allowed).

infile — is a specification of a file to be input to the new library file. If no infiles are supplied, an empty library file is created as the qualifiers dictate.

The operation qualifiers are as follows:

Qualifier	Description	Default
/SIZE:n	The size in 256-word blocks of the library file to be created.	100

/EPT:n	The number of entries to allocate in the entry point table (not greater than 1024). A macro library has no entry point table. n is rounded up to the nearest multiple of 64.	512(object) 0 (macro)
/MNT:n	The number of entries to allocate in the module name table (not greater than 1024). n is rounded up to the nearest multiple of 64.	256
/TYPE:type	The type of library being created. type is either OBJECT or MACRO.	OBJECT
/SELECT	The LINK command will use the file to define required global symbols at task build. (Object files only.)	
/SQUEEZE	Reduce the macro file by erasing all trailing blanks and tabs, blank lines and comments from the source text. (Macro files only).	

Examples

1. PDS> LIBRARIAN
   OPERATION? CREATE/SI:200/E:1024/M:512/TYPE:OBJ
   LIBRARY? MYLIB.OLB
   FILE? ONE.OBJ, TWO.OBJ, THREE.OBJ

   Create an object library file named MYLIB.OBJ with a size of 200 blocks with 1024 EPT entries and with 512 MNT entries, from three input files.

2. $LIBRARIAN CREATE/TYPE:MAC BATMAC.MAC INPUT.MAC

   Create a macro library file named BATLIB.MAC from one input file (INPUT.MAC).

DELETE

The DELETE operation performs two kinds of deletion:

1. It deletes modules, and all their associated entry points, from the library file specified.

2. It deletes specified entries in the entry point table (EPT).

Up to 15 modules may be deleted in one DELETE operation. If no module of the specified name exists in the library, DELETE has no effect on the library. A deleted module is marked as deleted, but remains physically in the file until a COMPRESS operation is performed.

Format

```
PDS>LIBRARIAN
OPERATION? DELETEqualifier
LIBRARY? libspec
ENTRIES? name-1[,...name-n]
```

or

```
$LIBRARIAN DELETEqualifier libspec name-1[,...name-n]
```

where

libspec — is a specification of the library file that contains the modules or entry points to be deleted.

name — is a module name or the name of an entry in the entry point table.

qualifier — is one of the following:

Qualifier	Description
/MODULES	Delete the specified module (the default qualifier).
/GLOBAL	Delete the EPT entries specified.

Examples

1.
```
PDS> LIB DELETE/MODULES
LIBRARY? MYLIB.MLB
ENTRIES? NAMEA, NAMEB, NAMEC
```

Delete the macros NAMEA, NAMEB and NAMEC from the macro library file MYLIB.MLB.

2.
```
$LIBRARIAN DELETE/GLOBAL  MACLIB.OLB  NAMEX
```

Delete the EPT entry named NAMEX contained in the library file MACLIB.OLB.

INSERT

The INSERT operation inserts modules into the specified library file. Any number of input files are allowed and all input files are considered to contain concatenated object modules.

Format

```
PDS> LIBRARIAN
OPERATION? INSERT[qualifier]
LIBRARY? libspec
FILE? infile-1[,...infile-n]
```

or

$LIBRARIAN INSERT[qualifier] libspec infile-1[,...infile-n]

where

libspec — is a specification of the library file into which modules are to be inserted (no wild-cards allowed).

infile — is the specification of a file containing concatenated object modules to be inserted into libspec.

qualifier is one of the following:

/SELECT — The LINK command will use the file to define required global symbols at task buld. (Object files only.).

/SQUEEZE — Reduce the macro-file by eliminating all trailing blanks and tabs, blank lines and comments from the source text. (Macro files only).

Examples

1. PDS> LIBRA
   OPERATION? INSERT/SQUEEZE
   LIBRARY? MACLIB.MLB
   FILE? ONE.MAC, TWO.MAC

   Insert the modules contained in the files ONE.MAC and TWO.MAC into the library file name MACLIB.MLB.

2. $LIBRARIAN INSERT MYLIB.OLB, MODULE.OBJ
   Insert the modules contained in the file MODULE.OBJ into the library file named MYLIB.OLB.

LIST

The LIST operation causes a library file directory to be printed at an output device (TO: by default) or to be sent to an output file. The operation qualifier also determines the amount of detail contained in the directory. By default, the directory lists all the modules in the library.

FORMAT

PDS> LIBRARIA
OPERATION? LIST[qualifier]
LIBRARY? libspec

or

$LIBRARIAN LIST[qualifier] libspec

where

libspec — is the specification of the library file to be listed (no wild-cards allowed).

qualifier — is one of the following:

Qualifier	Description
/OUTPUT::outfile	Send the output to the specified file.
/ENTRIES	Produce a directory of all modules and list entry points for each.
/FULL	Produce a directory of all modules, giving full module descriptions: size, date of insertion and version.

Examples

1. PDS> LIBRARIAN LIST MYLIB.MLB

   List at the user's terminal a directory of all the modules contained in MYLIB.MLB.

2. $LIBRARIAN LIST/FULL/OUTPUT:LPØ: MACLIB.OLB

   List at the line printer a directory of all the modules and their descriptions contained in the library file MACLIB.OLB.

REPLACE

The REPLACE operation replaces old modules in the library with new modules of the same name. That is, a new module that has the same name as a module already contained in the library replaces the existing module. The old module is simply deleted.

Format

```
PDS> LIBRARIAN
OPERATION? REPLACE[qualifier]
LIBRARY? libspec
FILE? infile-1[,...infile-n]
```

or

```
$LIBRARIAN REPLACE[qualifier] libspec infile-1[,...infile-n]
```

where

libspec — is the specification of the library file containing the modules to be replaced (no wild-cards allowed).

infile — is the specification of a file containing the new modules (no wild-cards allowed)

qualifier is one of the following:

/SELECT	The LINK command will use the file to define required global symbols at task build. (Object files only).
/SQUEEZE	Reduce the macro file by eliminating all trailing blanks and tabs, blank lines and comments from the source text. (Macro files only.)

Examples

1. PDS> LIBRARIAN
   OPERATION? REPLACE
   LIBRARY? MACLIB.OLB
   FILE? NEWMOD.OBJ

   Replace modules in the file MACLIB.OLB with modules of the same name in the file NEWMOD.OBJ.

2. $LIBRARIAN REPLACE OLDLIB.OLB ONELIB.OBJ,TWOLIB.OBJ

   Replace modules in the file OLDLIB.OLB with modules of the same name in the files ONELIB.OBJ and TWOLIB.OBJ.

LINK
$LINK

The LINK command links object files (that is, compiled or assembled modules) to form an executable task and produces output as directed by command qualifiers.

The IAS Task Builder Reference Manual describes the Task Builder procedures and options in full; anyone using Task Builder options should first read the Task Builder manual.

FORMAT

```
PDS>LINK[qualifiers]

FILE? infile-1[file-qualifier][,...,infile-n]
```

or

```
$LINK[qualifiers]   infile-1[file-qualifier][,..,infile-n]
```

where

infile — is the specification of an input file. See the section called Input Files below for further information.

Wild-cards are not allowed.

The user must not include this parameter if the command qualifier /OVERLAY has been specified (see the section called Command Qualifiers below)

file-qualifier — is one of the following file qualifiers. See the section called File Qualifiers for a definition of each qualifier.

/CONCATENATED

/LIBRARY

/LIBRARY:[(]mod-1[,...,mod-n)]

/NOCONCATENATED

/SELECT

qualifier(s) — are one or more of the command qualifiers listed below. The section called Command Qualifiers describes each one in detail.

Qualifier	Default
/ABORT	/ABORT
/CHECKPOINT	/CHECKPOINT
/DEBUG[:filespec]	/NODEBUG
/DISABLE	/DISABLE
/EXIT:n	/EXIT:1
/FIX	/FIX
/FLOATINGPOINT	/FLOATINGPOINT
/HEADER	/HEADER
/MAP[:filespec]	/NOMAP
/MULTIUSER	/NOMULTIUSER
/OPTIONS	/NOOPTIONS
/OVERLAY:filespec	/NOOVERLAY
/POSITIONINDEPENDENT	/NOPOSITIONINDEPENDENT
/PRIVILEGE	/NOPRIVILEGE
/SEQUENTIAL	/NOSEQUENTIAL
/SYMBOLS[:filespec]	/NOSYMBOLS
/TASK[:filespec]	/TASK
/TRACE	/NOTRACE

Command Qualifiers

All the command qualifiers described in this section may be negated by the prefix NO. For example, the qualifier /TASK instructs the Task Builder to keep a task file; whereas the qualifier /NOTASK requests that a task file not be kept.

/TASK[:filespec]

Default: /TASK

Keep a task image file.

Unless filespec is given, the task file takes the name of the first input file except that the extension is TSK.

If filespec is given, the extension field may be omitted; in which case, the Task Builder assumes it to be TSK.

If the command qualifier /OVERLAY is specified, filespec must be provided to name the task file.

/MAP[:filespec] or /MAP[:(filespec/SHORT)]

Default: /NOMAP

Produce a memory allocation map.

If filespec is not included with the /MAP qualifier, the map file is sent to the line printer.

If filespec is given, the extension field may be omitted; in which case, the Task Builder assumes it to be MAP.

If the command qualifier /OVERLAY is specified, filespec must be provided to name the map file.

The optional file qualifier /SHORT requests the Task Builder to produce a short map.

/SYMBOLS[:filespec]

Default: /NOSYMBOLS

Produce a symbol table file.

Unless filespec is given, the symbol table file takes the name of the first input file, except that the extension is STB.

If filespec is given, the extension field may be omitted; in which case, the Task Builder assumes it to be STB.

/OPTIONS

Default: /NOOPTIONS

Apply Task Builder options specified after the command string.

In interactive mode, the /OPTIONS qualifier causes the Task Builder to prompt "OPTIONS?" after the input files have been specified.

For example:

```
PDS> LINK/OPTIONS

FILE?  PROG REPORT

OPTIONS?
```

The user then enters the options which are described in the list below. A slash (/) as the first character in a line then terminates the list of options and the Task Builder begins executing. Details of individual option syntax are contained in the IAS Task Builder Reference Manual.

For example:

```
PDS> LINK/OPTIONS

FILES?  MAIN.OBJ, PROG.OBJ

OPTIONS?  ACTFIL=8

OPTIONS?  MAXBUF=160

OPTIONS?  UNITS=9

OPTIONS?  ASG=DT1:7:8:9

OPTIONS?  /
```

In batch mode, the presence of the /OPTIONS qualifier in the command qualifier list causes the Task Builder to expect one or more options to be specified on lines immediately following the command string.

A line containing a slash (/) in the first character position terminates the list of options.

The letters F and M in the list of Task Builder options below indicate for which language, FORTRAN or MACRO, the option is relevant.

Option	Meaning	
ABSPAT	Declare absolute patch values.	M
ACTFIL	Declare number of files open simultaneously.	FM
ASG	Declare device assignment to logical units.	FM
BASE	Define lowest virtual address.	FM
COMMON	Declare task's intention to access a shareable global area.	FM
EXTSCT	Declare extension of a program section.	FM
EXTTSK	Extend task memory allocation at install time.	FM
FMTBUF	Declare extension to buffer used for processing format strings at run-time.	F
GBLDEF	Declare a global symbol definition.	M

Option	Meaning	
GBLPAT	Declare a series of patch values relative to a global symbol.	M
LIBR	Declare task's intention to access a shareable global area.	FM
MAXBUF	Declare an extension to the FORTRAN record buffer.	F
ODTV	Declare the address and size of the debugging aid SST vector.	M
PAR	Declare partition name and dimensions.	FM
POOL	Declare pool usage limit.	FM
PRI	Declare priority.	FM
STACK	Declare the size of the stack.	FM
TASK	Declare the name of the task.	FM
TOP	Define highest virtual address.	FM
TSVK	Declare the address of the task SST vector.	M
UIC	Declare the user identification code under which the task runs.	FM
UNITS	Declare the maximum number of logical units.	FM

/OVERLAY:filespec

Default: /NOOVERLAY

Link the task according to the overlay structure defined in filespec, which must be included with the /OVERLAY qualifier. If the extension field of filespec is omitted, the Task Builder assumes it to be ODL.

The input files to LINK are specified within the overlay description file; therefore they must not be specified in the input file parameter list.

/DEBUG[:filespec]

Default: /NODEBUG

If filespec is not given, link the task with the system's debugging aid.

If filespec is given, link the task with the debugging aid contained in the specified file. The debugging aid must be in object format.

/ABORT

Default: /ABORT

The task can be aborted.

/CHECKPOINT

Default: /CHECKPOINT

The task can be checkpointed.

/DISABLE

Default: /DISABLE

The task can be disabled.

/EXIT:n

Default: /EXIT:1

Task Builder stops executing after n(decimal) errors.

/FIX

Default: /FIX

The task can be fixed in memory.

/FLOATINGPOINT

Default: /FLOATINGPOINT

The task uses the floating point processor.

/HEADER

Default: /HEADER

The task includes a header.

/MULTIUSER

Default: /NOMULTIUSER

The task is multiuser.

/POSITIONINDEPENDENT

Default: /NOPOSITIONINDEPENDENT

The task code is position independent.

/PRIVILEGE

Default: /NOPRIVILEGE

The task has privileged access rights.

/SEQUENTIAL

Default: /NOSEQUENTIAL

Program sections within the task are to be linked in the order in which they first appear. Otherwise they are linked in alphabetical order.

/TRACE

Default: /NOTRACE

The task is traceable.

Input Files

Input files to the LINK command may be specified in one of two ways:

1. In a list of file specifications as a parameter to the command.

2. From within an overlay description file by means of the /OVERLAY command qualifier.

If the /OVERLAY qualifier has been used to specify the input files, they must not also be specified as a command parameter (see item 1 above).

The input files may consist of:

1. Single object modules
2. Concatenated object modules
3. Object module libraries
4. Symbol table files

File qualifiers must be used to identify concatenated module files and library files (see the section called File Qualifiers below). In addition, the /SELECT qualifier may modify any type of object file; the Task Builder then uses the modified file only to resolve required symbol definitions.

The Task Builder provides default extensions in the following cases. When specifying single or concatenated object modules, the user may omit the extension field. The Task Builder then assumes the extension to be .OBJ. The extension field of a library file (a file modified by the /LIBRARY qualifier) may also be omitted, in which case the Task Builder assumes the extension to be OLB.

Symbol table files, however, have no default extension, so the extension field must be supplied.

Wild-cards are not allowed for any type of file specification supplied with LINK.

File Qualifiers - The following list defines all the available file qualifiers.

File Qualifier	Description
/CONCATENATED	Identifies the file as a concatenated object file.
/LIBRARY	Identifies the file as an object module library file.
/LIBRARY:[(]mod-1[,...,mod-n)]	Identifies the file as an object module library file where mod is the name of an object module and instructs the Task Builder to take only the modules named.
/NOCONCATENATED	Instructs the Task Builder to take only the first module in the file. If it is a concatenated object module file, subsequent modules are ignored.
/SELECT	Instructs the Task Builder to take only required global symbol definitions from the file. The modified file may be any object file, but it is normally a symbol table file.

EXAMPLES

1. $LINK/OPTIONS/PRIVILEGE A.OBJ/CONCATENATED
   UNITS=9
   /

2. PDS> LINK/OVERLAY:STRUCTURE/MAP:ROUTE

   The system does not prompt FILE? if /OVERLAY has been specified.

3. PDS> LINK

   FILE? A.OBJ, B.OBJ

LOGIN

The LOGIN command initiates an interactive session at a terminal.

FORMAT

PDS> LOGIN

USERID? username

PASSWORD? password

where

username is an alphanumeric character string 1 to 12 characters long which is unique to the user.

password is an alphanumeric character string 1 to 6 characters long associated with the user's username. As a security measure, the system does not print the password when it is entered in response to the PASSWORD? prompt.

DESCRIPTION

The LOGIN command is usually the first command issued by the interactive user (after the initial CTRL/C).

EXAMPLES

1. PDS> LOGI JOHNDOE

   PASSWORD? secret

   PDS>

2. PDS> LOGIN

   USERID? FREAN

   PASSWORD? cathy

   PDS>

LOGOUT

The LOGOUT command terminates the user's interactive session and releases any allocated devices and mounted volumes.

FORMAT

PDS> LOGOUT

The LOGOUT command has no parameters.

DESCRIPTION

The LOGOUT command causes the system to display the following information if "QUIET" mode has not been set:

1. The volumes and devices deallocated and dismounted
2. The user's username
3. The logout time
4. The connect time
5. CPU utilization

The message BYE then appears to indicate that the terminal is inactive.

EXAMPLE

PDS> LOGOUT

BYE

MACRO
$MACRO

The MACRO command assembles one or more ASCII source files containing MACRO-11 statements into a single relocatable binary object file. The output optionally consists of a binary object file and an assembly listing followed by the symbol table listing.

FORMAT

```
PDS> MACRO[qualifiers]

FILE? filespec[qualifier][+...+filespec]
```

or

```
$MACRO[qualifiers]  filespec[qualifiers][+...+filespec]
```

where

filespec is the specification of a file that contains MACRO source code. Multiple input file specifications must be concatenated with a plus sign (+). No wild-cards are allowed. Specifications must include a filename. If the extension is omitted, the system assumes it to be MAC.

qualifiers to the command are one or more of the following:

Qualifier	Meaning
/OBJECT[:filespec]	Produce an object file (the default condition), named accordingly if filespec (no wild-cards) is supplied. Otherwise the file is named by default (see Defaults below).
/NOOBJECT	Do not produce an object file.
/LIST[:filespec]	Produce a listing file (the default is /NOLIST in interactive mode and /LIST in batch mode), named accordingly if filespec is supplied. Otherwise the file is named by default (see Defaults below).
/NOLIST	Do not produce a listing file.

Defaults

Object File - By default the assembler produces an object file with the name of the last source file specified and the extension OBJ.

Listing File - A listing file is sent to the line printer by default in batch, or if /LIST specified in interactive mode.

COMMENTS

For further information on the use of MACRO-11, refer to the IAS/RSX-11 MACRO-11 Reference Manual.

EXAMPLES

1. PDS> MACRO/NOLIST
   FILE? A.AMC+B.MAC;3

2. $MACRO FILEA.MAC

3. PDS> MAC/OBJ:C.OBJ D.MAC+E.MAC

4. PDS> MAC MYFILE.MAC.

MESSAGE
$MESSAGE

The MESSAGE command sends a specified message to the operator's reporting terminal.

FORMAT

PDS> MESSAGE

MESSAGE? message

or

$MESSAGE message

where

message is a string terminated by carriage return in interactive mode, or

a string written on the same line as the $MESSAGE command in batch.

EXAMPLE

```
$MESSAGE THIS JOB WILL REQUIRE 2 TAPE DRIVES
```

MOUNT
$MOUNT

The MOUNT command makes a volume available to the user and optionally associates a logical name with the volume.

FORMAT

PDS> MOUNT[qualifier]

DEVICE? device-name

VOLUME-ID? volume-label

LOGICAL NAME? logical-name

or

$MOUNT[qualifier(s)] device-name volume-label logical-name

where

qualifier(s)	is one or more qualifiers, most of which may only be specified when a volume is initially mounted. See the section Command Qualifiers below.
device-name	is the device or the logical name of the device on which the volume is to be mounted. The device unit number may be omitted.
volume-label	is the volume identification written in the volume's header if the volume is labelled. If the volume does not have a label in its header, volume-label is the identification written on the volume container; in this case, /FOREIGN must be specified. For disk and DECtape the volume identification is 1 to 12 characters long. For ANSI labelled magnetic tape the volume label is 1 to 6 characters long.
logical-name	is the logical name to be associated with the device.

DESCRIPTION

The user obtains exclusive access to magnetic tape volumes and to any volumes mounted as foreign. Files-11 disk and DECtape volumes may be shared; that is, once the volume has been mounted, other users may also use it.

The unit number will normally be omitted from the device specification. The system then selects the appropriate unit.

The MOUNT command may be qualified in the following circmstances:

1. When a specified Files-11 disk or DECtape volume is not already mounted in the system.

2. When the user mounts a magnetic tape or foreign volume.

Command Qualifiers

The system ignores command qualifiers other than GLOBAL and REALTIME if the command is mounting a previously mounted Files-11 disk or DECtape.

* Qualifiers marked with an asterisk allow the user to override parameters set when the volume was initialized.

Qualifier	Description
*/ACCESSED:n	Number of preaccessed directories to be kept (Files-11 disk and DECtape only).
*/DENSITY:n	Set magnetic tape density where n = 800 or 1600
*/EXTENSION:n	Set file extension size to n blocks (Files-11 disk and DECtape volumes only)
*/FILEPROTECTION:(code)	Override default protection code to be given to new files. (See Chapter 6, section 6.1.3)
/FOREIGN	The volume to be mounted is foreign. The default is Files-11 format.
/GLOBAL	Volume to be mounted for access by any user. May not be specified for magnetic tape.
/NOWRITE	Write protected; that is, the volume may not be written to. Default is write permitted.
/OVERRIDE:(items)	where items are one or more of the following. Parentheses may be omitted if only one item is specified.
	EXPIRATION — allows the user to over-write an unexpired volume.
	SETID — allows the user to process tapes with inconsistent file set identifiers.
	VOLUMEID — allows the user to override the volume identification, thus allowing the user to mount without specifying a label.

Qualifier	Description
/PROTECTION:(code)	Replace volume protection with code specified. (See Chapter 6, section 6.1.3)
/REALTIME	The volume to be mounted is for Realtime use only
/UNLOCKED	Leave index file unlocked (Files-11 disk and DECtape only). Default is to leave index file locked.

EXAMPLES

1. PDS> MOUNT
   DEVICE? DT2:
   VOLUMEID? RISE <CR>

2. $MOUNT/FOREIGN MT: TESTER CFØ:

3. PDS> MOU DK:
   VOLUMEID? SAM AL1:

4. $MOUNT/DEN:8ØØ MT: VOL163 TAØ:

PASSWORD

The PASSWORD command changes the user's password.

FORMAT

```
PDS> PASSWORD
OLD PASSWORD? oldpassword
NEW PASSWORD? newpassword
```

where

oldpassword	is the 1- to 6-character alphanumeric password currently associated with the user's username.
newpassword	is the 1- to 6-character alphanumeric password that supersedes the old password.

DESCRIPTION

The system does not display either the old or the new password.

EXAMPLE

```
PDS> PASSWORD
OLD PASSWORD? glove
NEW PASSWORD? mitten
```

PRINT
$PRINT

The PRINT command causes one or more specified files to be output on the line printer. The user may optionally delete the file or files after they have been printed.

FORMAT

PDS> PRINT[/DELETE][/FORMS:n][/COPIES:n]

FILE? filespec-1[,...filespec-n]

or

$PRINT[/DELETE][/FORMS:n][/COPIES: n] filespec-1[,...filespec-n]

where

/DELETE instructs the system to delete the file or files after they have been printed.

/FORMS:n (where n is a digit from Ø to 6) indicates the type of form on which the specified files are to be printed.

Default: /FORMS:Ø

/COPIES:n (where n is an integer from 1 to 32) determines the number of file copies to be printed.

Default: /COPIES:1

filespec is the specification of a file to be printed. Wild-cards are allowed.

DESCRIPTION

The specified file or files are submitted to the line printer and subsequently deleted if the user has included the /DELETE qualifier.

EXAMPLES

1. PDS> PRINT

   FILES? A.EXT;4

2. $PRINT FREAN.MAC;3, PEEK.CAF;*

3. PDS> PRI/DE B4.FAL;1

PROTECT
$PROTECT

The PROTECT command applies a new protection code to a specified file.

FORMAT

PDS> PROTECT

FILE? filespec

PROTECTION? (code)

or

$PROTECT filespec (code)

where

filespec is the specification of the file to which the protection code is to be applied.

(code) is the protection code to be applied to filespec. See Chapter 6, section 6.1.3.

User categories are:

SYSTEM:

OWNER:

GROUP:

WORLD:

Types of access are:

R read

W write

E extend

D delete

Example

(SY:R, O:RWED, GRO:RW)

System has read access only. Owner has all four types of access. Group has read and write access only. World access remains unchanged.

EXAMPLES

1. PDS> PROTECT CATHS.DAT

   PROTECTION? (GRO:R, SY:R, WORLD:, O:RWDE)

2. $PROTECT TONY.MAC (OW:RWED, SY:, GR:, W:)

3. PDS> PROTECT MYPROG.COB (SY:RWED, OW:RWDE, WO:DERW, GR:RWED)

QUEUE

The QUEUE command allows the user to access the queue in three ways:

1. To interrogate the queue (/LIST)
2. To remove an entry from the queue (/REMOVE)
3. To add to the queue (/ADD)

Note that the simpler commands PRINT and SUBMIT should be used to add files to the line printer and batch queues.

FORMAT

The format of the command depends on the queue operation to be performed.

The default operation is /ADD.

LIST

PDS> QUEUE/LIST

Description

Display the status of the user's queue entries.

REMOVE

PDS> QUEUE/REMOVE

SEQUENCE? seqno

where

seqno is the sequence number of a queue entry to be removed, determined by issuing a QUEUE/LIST command.

Description

Remove the queue entry specified by a sequence number.

ADD

PDS> QUEUE/ADD[/FORMS:n][/COPIES:n][/DELETE]

QUEUE? device-name

FILE? filespec

where

/FORMS:n	(where n is a digit from 0 to 6) indicates the type of form on which the specified files are to be printed. Default: /FORMS:0
/COPIES:n	(where n is an integer from 1 to 32) determines the number of copies to be printed. Default: /COPIES:1
/DELETE	requests the system to delete the specified files after they have been processed.
device-name	specifies the relevant queued device.
filespec	is the specification of a file to be added to the queue specified. Only one filespec is allowed. It must contain a filename and an extension. Wild-cards are allowed.

Description

Add the specified file to the named queue and, optionally, modify the resultant operation according to any specified qualifiers.

EXAMPLES

1. PDS> QUEUE/LIST

2. PDS> QUEUE/REMOVE 2

3. PDS> QUEUE/ADD/COPIES:4/DELETE
   QUEUE? LP0:
   FILE? LIST.MAP;4

RENAME
$RENAME

The RENAME command renames an existing file.

FORMAT

```
PDS> RENAME
OLD? oldspec
NEW? newspec
```

or

```
$RENAME oldspec, newspec
```

where

oldspec is the specification of an existing file.

newspec is the new name for oldspec.

DESCRIPTION

Both oldspec and newspec must contain a file name and an extension. Wild-cards are allowed. The device field in both file specifications must be the same because files cannot be renamed from one device to another. If the version field is omitted, the normal defaults apply (see Chapter 6, section 6.2.1).

EXAMPLES

1.
```
PDS> RENAME
OLD? MYFILE.OBJ;1
NEW? BACKUP.OBJ;1
```

2.
```
$RENAME MYFILE.OBJ;1,BACKUP.OBJ;1
```

3.
```
PDS> RENAME
OLD? MYFILE.OBJ;1,BACKUP.OBJ;1
```

4.
```
PDS> RENAME CAROL.*;*
NEW? FRED.CBL;*
```

RUN
$RUN

The RUN command causes an executable task to execute.

FORMAT

PDS> RUN

FILE? filespec[/COBOL]

or

$RUN filespec[/COBOL]

where

filespec is the specification of a file that contains an executable task. The specification must include a file name. If the extension is omitted, TSK is assumed, unless /COBOL is specified. The default extension for a COBOL specification is OBJ.

/COBOL Declares that filespec contains a compiled COBOL program.

DESCRIPTION

To run a COBOL program, users must include the file qualifier /COBOL.

To suspend an executing task, the user types CTRL/C. The user may either type CONTINUE to resume task execution or ABORT to abort the task.

Executing tasks that were submitted to the batch queue cannot be suspended.

EXAMPLES

1. PDS> RUN MYPROG
2. $RUN MYPROG
3. $RUN COBOL.OBJ/COBOL

SET
$SET

The SET command either:

1. Establishes a new default device or UFD or both for subsequent file specifications supplied by the user (SET DEFAULT), or
2. Suppresses the output of certain information messages (SET QUIET)

FORMAT

PDS> SET

FUNCTION? parameter

or

$SET parameter

where

parameter is one of the following:

QUIET or NOQUIET	Suppress or allow the output of informative (usually accounting) messages. The system default is SET NOQUIET.
DEFAULT [device-name][ufd]	Change the user's default device and/or UFD to the value or values specified. If both device-name and ufd are omitted, the system reestablishes the user's initial default settings for both values.

DESCRIPTION

Changing Defaults

The system manager allocates a default device to each user, which is in effect when the user logs in. The initial default UFD is equivalent to the user's UIC. The user must issue the SET DEFAULT command to change either or both values for file specifications

included in subsequent commands. The command does not affect file specifications written in programs. To re-establish the default settings in effect at login, the user issues SET DEFAULT without any other values.

EXAMPLES

1. PDS> SET QUIET DK1:[200,3]

2. $SET DEFAULT [30,3]

3. PDS> SET DEFAULT DK0:

SHOW

The SHOW command causes the terminal to display specified information at the user's terminal. The parameter to SHOW determines the type of information displayed.

FORMAT

PDS> SHOW

ATTRIBUTE? parameter

where

parameter specifies the type of information to be displayed. The options are:

CLI	Display information about the Command Language Interpreters (CLIs) currently running in the system.
DEFAULT	Display the user's current default device and UFD
DEVICES	Display information about all the devices known to the system. See the section called Devices below.
TIME	Display the current time and date.
STATUS	Display information about the current status of the user's job.

Devices

The command SHOW DEVICES causes the system to display at the user's terminal the symbolic names of all the devices known to the system. Physical device names are followed by "**" if they are currently available for use. System logical device names are followed by the associated physical device names. The listing also includes messages giving additional information about particular devices. The messages and their meanings are:

Message	Meaning
GLOBAL	The device has been mounted globally (see the MOUNT command).
MOUNTED	The device is mounted.
REALTIME	The device is mounted for realtime activity.

Message	Meaning
SPOOLED	The device is spooled.
TIMESHARING:n	n is the number of timesharing users accessing the device.

EXAMPLES

1. PDS> SHOW TIME

   10:53:41 1-OCT-75

2. PDS> SHOW DEVICES

```
TT0       **
CI0       TT0
CO0       TT0
CL0       LP0
SP0       DB0
PI0       **
MO0
MM0
DT1       **
DT0       ** MOUNTED GLOBAL
LP0       ** SPOOLED
TT10      **
TT7       **
TT6       **
TT5       **
TT4       **
TT3       **
TT2       **
TT1       **
DB0       ** MOUNTED GLOBAL TIMESHARING:4
DK1       **
DK0       ** MOUNTED GLOBAL
SY0       DB0
```

SUBMIT

The SUBMIT command sends a file containing batch commands to the batch processor.

FORMAT

PDS> SUBMIT

FILE? filespec

where

filespec is the specification of a file containing batch commands. The specification must contain a filename and an extension.

DESCRIPTION

The system submits the file of batch commands to a queue of jobs to be run eventually by a batch processor activated to service queued input.

EXAMPLES

1. PDS> SUBMIT

   FILE? BATCHFILE.CMD

2. PDS> SUBMIT BATCHFILE.CMD

3. PDS> SUBMIT MYJOB.CMD

TYPE

The TYPE command causes the contents of one or more specified files to be printed at the user's terminal.

FORMAT

```
PDS> TYPE

FILE? filespec-1[,...filespec-n]
```

where

filespec is the specification of a file. The specification must contain a filename and an extension. Wild-cards are allowed.

EXAMPLES

1.
```
PDS> TYPE

FILE? (BARLEY.CBL;2, GRAHAM.CBL;2)
```

2.
```
PDS> TYPE APPLE.DAT
```

UNLOCK
$UNLOCK

The UNLOCK command unlocks a file that was locked as a result of being improperly closed.

FORMAT

```
PDS> UNLOCK

FILE? filespec-1[,...,filespec-n]
```

or

```
$UNLOCK filespec-1[,...,filespec-n]
```

where

filespec is the specification of the file to be unlocked. Wild-cards are not allowed.

DESCRIPTION

If a program using File Control Services (FCS) has a file open with write access and exits without first closing the file, the file will be locked against further access as a warning that it may not contain proper information. Typically the following information would not have been written to the file:

1. The current block buffer being altered.
2. The record attributes which contain the end of file information.

By using the UNLOCK command, the user can access the file and can determine the extent of the damage and perhaps take appropriate corrective action.

EXAMPLE

```
PDS> UNLOCK

FILE? THAMES.MAC;7
```

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Communications
P. O. Box F
Maynard, Massachusetts 01754

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form.

Did you find errors in this manual? If so, specify by page.

_____

_____

_____

_____

_____

_____

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

_____

_____

_____

_____

_____

_____

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

_____

_____

_____

_____

_____

_____

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

If you require a written reply, please check here. ☐

Please cut along this line.