RSX-11M-PLUS
Indirect Command Processor
Manual

Order No. AA-JS10A-TC

RSX-11M-PLUS Version 4.0

Digital Equipment Corporation Maynard, Massachusetts

First Printing, July 1985
Revised, August 1987

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

The following are trademarks of Digital Equipment Corporation:

DEC	EduSystem	UNIBUS
DEC/CMS	IAS	VAX
DEC/MMS	MASSBUS	VAXcluster
DECnet	MicroPDP-11	VMS
DECsystem-10	Micro/RSX	VT
DECSYSTEM-20	PDP	
DECUS	PDT	
DECwriter	RSTS	digital
DIBOL	RSX	

ZK3080

HOW TO ORDER ADDITIONAL DOCUMENTATION
DIRECT MAIL ORDERS

USA & PUERTO RICO*	CANADA	INTERNATIONAL
Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061	Digital Equipment of Canada Ltd. 100 Herzberg Road Kanata, Ontario K2K 2A6 Attn: Direct Order Desk	Digital Equipment Corporation PSG Business Manager c/o Digital's local subsidiary or approved distributor

In Continental USA and Puerto Rico call 800-258-1710.

In New Hampshire, Alaska, and Hawaii call 603-884-6660.

In Canada call 800-267-6215.

*Any prepaid order from Puerto Rico must be placed with the local Digital subsidiary (809-754-7575).

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Westminster, Massachusetts 01473.

This document was prepared using an in-house documentation production system. All page composition and make-up was performed by TEX, the typesetting system developed by Donald E. Knuth at Stanford University. TEX is a trademark of the American Mathematical Society.

Contents

Preface

Manual Objectives

The *RSX-11M-PLUS Indirect Command Processor Manual* describes Indirect, the task used to run indirect command files and to perform other programming and system-control functions. The manual discusses the different kinds of indirect command files and their uses, and describes Indirect's directives and special symbols and how to use them.

Intended Audience

This manual is intended for anyone who is interested in learning about the Indirect Command Processor and how to use it.

Structure of This Document

A Summary of Technical Changes provides information about changes to, and new features of, the RSX-11M-PLUS operating system that affect Indirect.

Chapter 1 is an introduction to Indirect. It explains what Indirect is and gives an overview of the various features of Indirect. Examples at the end of the chapter illustrate different ways in which to use Indirect.

Chapter 2 is a reference section on Indirect. It explains in more detail the functions of Indirect and its directives and symbols. More examples appear at the end of this chapter.

Chapter 3 describes the Indirect Pre-processor (IPP) and provides tables showing the short forms for all Indirect directives and special symbols.

Appendix A lists and explains all of the Indirect messages.

Associated Documents

Other RSX-11M-PLUS system manuals that supplement this manual are the *RSX-11M-PLUS MCR Operations Manual* and the *RSX-11M-PLUS Command Language Manual*. These manuals give more detail about the way the system operates and describe the commands mentioned in this manual.

Conventions Used in This Document

A number of conventions are used in the directive descriptions in this manual:

Convention	Meaning
. . .	The vertical ellipsis shows where elements of command input have been omitted because they are not relevant to the point being discussed.
[parameter]	Any command parameter enclosed in square brackets is optional. If the brackets include syntactical elements, such as periods (.) or slashes (/), those elements are required for the parameter. If the parameter appears in lowercase, you are to substitute a valid command element if you include the parameter.
[g,m]	This signifies a User Identification Code (UIC). The g is a group number and m is a member number. The UIC identifies a user and is used mainly for controlling access to files and privileged system functions. This sometimes also signifies a User File Directory (UFD), commonly called a directory. Where a directory name is required, only one set of brackets is shown, as in [g,m]. Where the directory is optional, two sets of brackets are shown, as in [[g,m]]. Other notations for directories are [ggg,mmm], [gggmmm], [ufd], [name], and [directory].
UPPERCASE	Any command parameter in uppercase indicates the valid form of the command. If you type it in that form, it will work as described.
lowercase	Any command parameter in lowercase is to be substituted for. Usually, the lowercase word identifies the kind of substitution expected, such as filespec, which indicates that you should fill in a file specification.
/switch	Switches alter the action of the directive to which they are attached.
parameter	Required command fields are generally called parameters. The most common parameters are file specifications.
filespec	A full file specification includes device, directory, file name, file type, and version number, as in this example: `DL2:[46,63]INDIRECT.TXT;3` Full file specifications are rarely needed. If you do not provide a version number, the highest numbered version is used. If you do not provide a directory, the default directory is used. Some system functions default to particular file types.
red ink	All user-input text in examples is printed in red ink to distinguish it from system output. That is, what you type is shown in red.

Convention	Meaning
[RET]	A rectangular symbol with a 2- to 6-character abbreviation indicates that you are to press the corresponding key on your terminal. For example, [RET] indicates that you are to press the RETURN key and [DEL] means that you are to press the DELETE key.
[CTRL/a]	The rectangular symbol [CTRL/a] means that you are to press the key marked CTRL while pressing another key. Thus, [CTRL/Z] indicates that you are to press the CTRL key and the Z key simultaneously. [CTRL/Z] is echoed on your terminal as ^Z, but not all control characters echo.

Summary of Technical Changes

This edition of the *RSX-11M-PLUS Indirect Command Processor Manual* contains changes and additions made to the RSX-11M-PLUS Version 4.0 operating system. These changes and additions are as follows:

- A new feature, the Indirect Pre-Processor (IPP), reads an indirect command file, changes all directives and special symbol notations to a shortened form, and writes a new file that has the shortened command lines in it. The file has the type CMF.
- The .ERASE SYMBOL directive can now be used to delete local symbols.
- The following error message is new:

  ```
  .EXIT without .END
  ```

  The following error message is not new, but has never been fully documented before:

  ```
  File attributes not available
  ```

- Two new special symbols are described in this manual:

<REGSEG>	Assigned the number, in octal, of groups of free bytes in the internal string symbol storage region. This symbol can be used to detect severe fragmentation in the region, which occurs because the region is not shuffled after each addition or deletion of a variable.
<REGSIZ>	Assigned the number, in octal, of free bytes in the internal string storage region.

Chapter 1
Introduction to Indirect

What is Indirect? Indirect is a command processor that saves you time and energy by doing a lot of work on the system for you. It also reduces the frustration that results from inevitable typing mistakes.

Why is it called "Indirect"? Because it changes the way you interact with the system from one of immediate user action/system reaction — you type out and enter a command, the system executes it and waits for another one — to an *indirect* interaction between you and the system. The Indirect Command Processor allows you to put the commands in a file and tell the system to execute them while you do something else. Instead of entering commands directly to the system, you provide an indirect reference to the file that has all the commands in it.

Indirect also has its own directives and symbols with which you can create programs to do a variety of tasks. Indirect runs from a logged-in terminal and always runs at the same priority.

The following sections describe more about Indirect.

Indirect Command Processing

Indirect lets you execute one or more command line interpreter (CLI) commands by typing one Indirect command line.

You create a file, and put the CLI commands you want to execute into the file in the order you want them processed. To execute this command file, type an at sign (@) and the name of the file. Indirect and the CLI then do all the work.

For example, the command file EXAMPLE.CMD contains the following DCL command lines:

```
SHOW TIME
DIRECTORY WORKLIST.TSK;*
RUN WORKLIST
PRINT WORKLIST.MAP,WORKLIST.LST
SHOW TIME
```

To execute this command file, type the following command line:

```
$ @EXAMPLE RET
```

Indirect (invoked by the at sign) reads the command lines in the file one line at a time, waiting until each command has been executed before going on to the next one.

Use an editor (such as EDT) to create your command file. Because Indirect looks for CMD file types by default (or CMF file types if you have pre-processed your CMD file; see Chapter 3 for more information), you should create your file with this file type. If you name it something else, you must specify the different file type when you execute the file.

Indirect accepts input in both uppercase and lowercase characters. When it prompts you for information, it displays the question exactly as it was put into the file.

Indirect Command Files

Indirect command files are used for many different things. One example is a login command file. When you log in, the system automatically runs the LOGIN.CMD file in your login directory, which can, for instance, set various characteristics for your terminal or automatically run other programs or files.

To illustrate, you can use LOGIN.CMD to change the characteristics of your terminal if the ones you want are different from the terminal's default characteristics. Put the necessary CLI command lines in LOGIN.CMD. The characteristics will be changed automatically when you log in. Here is an example of a DCL login command file:

```
SET TERMINAL/SPEED:(9600,9600)/WIDTH:80
@COOKIE
SHOW DEVICES
SHOW USERS
SHOW TIME
```

This command file sets two different characteristics for the terminal: speed and width. (See the *RSX-11M-PLUS Command Language Manual* for more information about these commands.) The command file also runs another command file, COOKIE.CMD. When COOKIE.CMD finishes executing, Indirect returns to the first file (the login command file) to continue executing it. The SHOW commands display on the terminal lists of the peripheral devices on the system (terminals, disk drives, line printers, and so on) and the users currently logged in on the system, and then the current time and date.

When you include commands in a login command file, you do not have to type those commands every time you log in; Indirect does all the work for you. Including repetitive sequences of commands that you are going to use often into a single file is something for which Indirect is especially helpful. Using indirect command files saves you time and prevents mistakes. "Repetitive sequences of commands" can be just about anything. A few examples are listing files in your directory, mounting volumes, backing up files, and doing quick tests at your terminal. The following sample DCL command files will give you a better idea of what Indirect can do for you:

- To prepare a disk volume for use and then mount it, run a file containing the following command lines:

  ```
  MOUNT/FOREIGN DU1:
  ANALYZE/MEDIA DU1:
  DISMOUNT DU1:
  INITIALIZE DU1:MYDISK
  MOUNT/NOSHAREABLE DU1:MYDISK
  ```

This file checks the volume (DU1:) for bad blocks (so that data will not be written to them), initializes the volume (which deletes any data currently on the volume and makes it a Files-11 formatted volume), and then mounts the volume. (Note that the device on which you mount a volume might not be DU1:, nor is the name of the volume likely to be "MYDISK.")

- To check for files in your directory, use a file containing command lines similar to the following:

```
DIR *.RNO;*
DIR *.MEM;*
DIR *.TXT;*
DIR *.LST;*
DIR *.CMD;*
DIR *.HLP;*
```

These command lines display on your terminal lists of various files based on their file type. After looking at these lists, you can decide what you want to do with the files.

Substitution Mode

You may need to change indirect command files often to make them do exactly what you want to do each time. For example, you might use a command file to do a backup procedure, but find that you have to edit the file to change the name of the device drive or its unit number. For such cases, Indirect has substitution mode.

Substitution mode allows you to place a special word — called a symbol — in the command line. When you run the command file, it will ask you (through a special Indirect command line you include in the file) for the information that is to be substituted for the symbol. An Indirect directive (or command), .ENABLE SUBSTITUTION, allows you to use substitution mode.

The following command file shows substitution mode being used:

```
.ENABLE SUBSTITUTION
.ASKS DEVICE Device to mount?
MOUNT 'DEVICE'
```

These command lines (which can be part of a larger command file) perform the following actions: they enable substitution mode, ask you which device is going to be mounted (DEVICE), and then mount that device. The apostrophes around DEVICE tell Indirect to take your answer to the system's question and substitute that value for DEVICE before it processes the command line. When you run the file, this is what you see on your terminal:

```
>* Device to mount? [S]: DU1: [RET]
>MOUNT DU1:
```

When you see "* Device to mount? [S]:" prompting you on your terminal, type in the name of the device to be mounted and then press the RETURN key. After you have answered the question, Indirect displays the MOUNT command line on your terminal, with the specific device name substituted for 'DEVICE,' and the system mounts the device.

For CLI commands and for questions displayed by the .ASKx directives, the first character displayed is the current CLI prompt. For MCR, the default prompt is the right angle bracket (>). For DCL, the default prompt is the dollar sign followed by a space ($).

The asterisk (*) at the beginning of the line indicates that the question is being asked by Indirect. .ASKS means "ask for a string," so the "[S]:" at the end of the question indicates that Indirect expects a string answer, that is, an answer containing a string of alphabetic and/or numeric characters. Indirect also accepts other types of answers, depending on the question being asked.

When the command file is executed, Indirect substitutes your answer to the question (DU1:) for the symbol 'DEVICE' in the MOUNT command line following the question. That is why you see "MOUNT DU1:" displayed on your terminal instead of "MOUNT 'DEVICE'." Using substitution mode lets you name any device with your command file.

The next (MCR) command file, which displays information from a user's local (that is, private) help file, shows a similar use for substitution mode. (These commands could also be part of a larger file.)

```
.ENABLE SUBSTITUTION
.ASKS CMND Enter command name
HELP/FIL:DU2:[303,23]COMMANDS.HLP 'CMND'
```

The terminal session would be this:

```
>@HELP RET
>* Enter command name [S]: Telegram RET
>HELP/FIL:DU2:[303,23]COMMANDS.HLP Telegram
  The TELEGRAM command sends a specified message . . .
```

As a way to display help files, this command file asks for the topic for which help is wanted. When the terminal displays "* Enter command name [S]:," you type in the topic you want help on. Indirect takes your answer, substitutes it for the symbol 'CMND' in the HELP command line, and then displays the requested help information immediately afterwards. (See the *RSX-11M-PLUS MCR Operations Manual* for information on the HELP command.)

Writing Programs with Indirect

As you can see, Indirect can be used to write programs—in fact, the command file shown above is really a simple program. Many common programming techniques are available in Indirect. These techniques include looping, counters, variables, arithmetic and logical operations, and testing system conditions. The techniques are performed through the use of Indirect directives, symbols, and labels.

Directives

.ENABLE SUBSTITUTION and .ASKS are only two of the many Indirect directives. This chapter will not describe all of the directives, but will acquaint you with a few that you are most likely to use and to use frequently. The .ASKS directive has two companion directives, .ASK (for true/false — or logical — questions) and .ASKN (for numeric questions). You can use .ENABLE and its companion directive, .DISABLE, to set and change several other modes in Indirect.

All Indirect directives begin with a period, except for the logical end-of-file directive, which is a slash (/).

For a complete list of the directives, see Chapter 2.

Special Symbols

Indirect has special symbols that it defines automatically. The symbols are dependent upon specific system characteristics and the replies to queries given during command file execution. Special symbols can be compared, tested, or substituted and are of three types: logical, numeric, or string. All special symbols have a common format: angle brackets (<>) enclose the special symbol name.

For a complete list of the special symbols, see Chapter 2.

Labels

You can also use labels in command files. Labels allow you to organize your file more coherently and to jump to other lines in the file, depending on the results of conditional statements. For example, the following command file asks for the values of two variables and then compares them.

```
        .ENABLE SUBSTITUTION
        .ASKN A Enter value for A
        .ASKN B Enter value for B
        .IF A > B .GOTO TEST2
        .EXIT
.TEST2: .SETN A B
            .
            .
            .
```

Depending on the result of the comparison (performed by the .IF directive), the command file either exits (.EXIT) or proceeds to the section of the file labeled .TEST2:.

Notice that the label begins in the first column of the command file while the directives begin in the ninth column (one tab stop over). Formatting your command files in this way makes them consistent and easy to read.

Labels are one to six characters in length, begin with a period (.), and end with a colon (:). (The period and colon are not included in the six characters.) When you use labels in command lines within the command file, however, you only need to use the name; you do not need to include the period and colon. The .GOTO directive allows you to go to the different sections of the file marked by different labels.

The .IF and .SET directives, like .ASKS, have companion directives. The other .IF directives allow you to make tests for certain specific conditions. The other .SET directives allow you to set values as true, false, logical, numeric, string, octal, or decimal.

The following command file uses one of the other .SET directives, .SETS, and also the .ENABLE and .GOTO directives. The file also uses the special string symbol <TIME> . A more detailed explanation follows the text of the file.

```
.; The following file prints a message on the terminal,
.; depending on the time of day.
.ENABLE SUBSTITUTION
.SETS TIME "'<TIME>'"
.; <TIME> has the format hh:mm:ss.

.SETS SAYING TIME[8.:8.]
.; Sets SAYING equal to last digit of <TIME> (1's column
.; for seconds).
```

```
        .GOTO 'SAYING'00
        .; Makes a label based on the second <TIME> is checked.
.000:
        ;  What else can go wrong?
        .GOTO END
.100:
        ;  Have you seen your shrink today?
        .GOTO END
.200:
        ;  Ours is not to reason why.
        .GOTO END
.300:
        ;  Where were YOU when the lights went out?
        .GOTO END
.400:
        ;  Why are you here?
        .GOTO END
.500:
        ;  Everything is relative.
        .GOTO END
.600:
        ;  It will be a good experience for you!
        .GOTO END
.700:
        ;  Don't panic.
        .GOTO END
.800:
        ;  One lousy driver can ruin your whole day.
        .GOTO END
.900:
        ;  Curiosity killed the cat.
        .GOTO END
.END:
        .EXIT
```

Explanation of the Command File

In addition to the directives and special symbol, this command file illustrates other features of Indirect. The first feature is the use of comments. Comments can be used to describe what the file is supposed to do, and to explain what the command lines do or to give additional information about them. Comments that begin with a period and semicolon (.;) will not be displayed on the terminal when the file is executed. Comments that begin with only a semicolon (;) or an exclamation point (!) will be displayed.

This file, as the introductory comment explains, displays a message on the terminal when the file is run. The message displayed depends on the time at which the file is executed.

When the file begins to execute, substitution mode is enabled and the symbol TIME is set with the .SETS directive to be equal to the contents of the special symbol <TIME>. <TIME> contains the current time in the format hh:mm:ss. The second .SETS command line sets the symbol SAYING to be equal to the last digit contained in <TIME>. The range [8.:8.] tells

Indirect to look for the last character in the string of eight characters; in other words, the second digit of seconds (ss). For example, if <TIME> contains 11:37:56, the symbol SAYING is set to 6. That means that Indirect will display the message:

```
;  It will be a good experience for you!
```

The .GOTO command line creates a label, using the second from <TIME>, so that Indirect will know which label to go to and which message to display. (In the above example, Indirect branched to label .600:.) The remainder of the file lists the labels and the messages to be displayed, and then branches to the .END: label after the message has been displayed. In that way, Indirect goes directly to the end of the file and exits (.EXIT) without first displaying any messages following the one that was displayed.

The following examples will give you more of an idea of the usefulness and versatility of Indirect. A brief commentary follows each example. For more information on Indirect (directives, symbols, switches, and so on), see Chapter 2.

Examples

An explanation of the example follows each one.

- The following DCL command file prepares a new diskette for use on your system:

```
; Place the new diskette in one of the drives before
; answering the question.
.ENABLE SUBSTITUTION
; Diskette drives are named DU1: and DU2:.
.ASKS DISK Which diskette drive
; Labels can have up to 12 letters and numbers.
.ASKS LABEL What label do you want
MOUNT/FOREIGN/NOSHAREABLE 'DISK'
ANALYZE/MEDIA 'DISK'
INITIALIZE 'DISK' 'LABEL'
DISMOUNT 'DISK'
MOUNT/NOSHAREABLE 'DISK' 'LABEL'
CREATE/DIRECTORY 'DISK'
; Diskette in 'DISK' is ready for use.
```

  In this file, you instruct the system to tell you to place the new diskette in the drive that you will be using. To have the system display this kind of information, include comments beginning with a semicolon (;) at the appropriate places in the command file. Comments that begin with a semicolon are always displayed. Comments that begin with a period and a semicolon (.;) are not displayed.

  The first command line in the file enables substitution mode. When you enable substitution mode, Indirect can substitute the value of a symbol in a command line or directive statement. The next line displays information about the diskette drives on the system. The .ASKS command line asks you which drive you will be using. In this example, you name the drive with the new diskette in it. Once you have answered the question, Indirect substitutes the name of the drive you specified wherever 'DISK' appears in a command line. (Remember that the apostrophes are required for the substitution operation to take place.) Although Indirect allows you to check for correct syntax, this sample command file does not take advantage of that option.

The next line displays information about labels, and the succeeding .ASKS command line asks you for the label of the diskette. The label is an identifier for the diskette volume and a password for using the diskette. No one can mount the diskette without knowing the label.

The MOUNT command mounts the diskette so that the system can work with it. The /FOREIGN qualifier is used because the volume is not yet formatted properly for use on an RSX-11M-PLUS system. The /NOSHAREABLE qualifier means that no one else can use the diskette while you are using it.

The ANALYZE command line tells the system to look for bad blocks on the diskette. Bad blocks are areas on the diskette volume that cannot be used for reading or writing data. If the system determines beforehand where the bad blocks are, it can avoid them during read and write operations to the diskette.

The INITIALIZE command reformats the diskette so that it is in Files-11 format. Files-11 is the standard RSX-11M-PLUS format for disk volumes. The DISMOUNT and second MOUNT commands are necessary after the diskette has been initialized because they inform the system that it can now treat the diskette as a standard Files-11 volume.

The CREATE/DIRECTORY command line creates a new directory on the diskette. The directory on the diskette is the same as your login directory on your user disk. The last line of the command file displays the statement that the specified diskette is now ready for you to use.

- The following command file concatenates several DCL help files into one file named HELPFILES.TXT and then prints the file after a certain specified time:

```
.ENABLE SUBSTITUTION
.ASKS DEVICE Enter device and directory spec
.ASKS TIME Time to print (hh:mm)?
.IF TIME EQ "" .SETS TIME "0"
COPY 'DEVICE'DCL.HLP HELPFILES.TXT
APPEND 'DEVICE'ALLOCATE.HLP,BROADCAST,COPY HELPFILES.TXT
APPEND 'DEVICE'DIRECTORY.HLP,DISMOUNT,HELP HELPFILES.TXT
APPEND 'DEVICE'INITIALIZE.HLP,LINK,MOUNT HELPFILES.TXT
APPEND 'DEVICE'PURGE.HLP,RENAME,RUN,SET HELPFILES.TXT
PRINT/AFTER:('TIME') HELPFILES.TXT
```

  In this file, substitution mode is enabled, and Indirect asks which device and directory the files are to be copied from ('DEVICE') and the time after which the files are to be printed ('TIME'). If no specific time is given, the files are queued to be printed immediately.

  The COPY command creates the new file HELPFILES.TXT and copies into it the help file DCL.HLP. The APPEND commands add more help files to the end of HELPFILES.TXT. The PRINT command prints HELPFILES.TXT after the time given in response to the "Time to print?" question.

- The following command file can help you delete unnecessary files from your directory:

```
        .ENABLE SUBSTITUTION
.BEGIN:
        .ASKS FILE Which file?
        TYPE 'FILE'
        .ASK DEL Delete this file
        .IFT DEL DELETE 'FILE';*
        .GOTO BEGIN
```

With this file, substitution mode is enabled and Indirect asks for the name of a file to be deleted. However, before the file is deleted, DCL (the CLI for this example) displays the file on the terminal and Indirect asks whether the file should be deleted. This verification ensures that you do not delete a file that you really want to keep.

If you answer "Yes" (Y) to the question, DCL deletes the file. After the file is deleted, Indirect loops back up to the beginning and asks for the name of the next file to be deleted. If you have no more files to be deleted, press CTRL/Z in response to the "* Which file?" question.

- The following command file gets information about the system, your account, and your terminal, and writes the information into another file:

```
.ENABLE SUBSTITUTION
.OPEN INFO.DAT

.ENABLE DATA
'<DATE>'            ! This is today''s date.

'<TIME>'            ! This is the current time.

'<UIC>'             ! This is your current UIC.

'<LOGDEV>'          ! This is your login device.

'<NETNOD>'          ! This is the DECnet node name of
                    ! your system.

'<TISPED>'          ! This is the baud-rate code for
                    ! your terminal.

'<TITYPE>'          ! This is the type code for the
                    ! terminal you are using.
.DISABLE DATA
.CLOSE INFO.DAT
TYPE INFO.DAT
```

With this file, substitution mode is enabled, a new file called INFO.DAT is opened so that the information can be written into it (if the file already exists, Indirect will create a new version), and then data mode is enabled. Data mode allows several lines of text to be written into a file.

Next, Indirect gets the contents of the various special symbols and writes the information into INFO.DAT. After the last symbol is read, data mode is disabled, and INFO.DAT is closed and then displayed on the terminal. For example:

```
$ @INFORM [RET]
$ TYPE INFO.DAT
14-JUL-87               ! This is today's date.

10:14:37                ! This is the current time.

[303,23]                ! This is your current UIC.
```

```
DUO                     ! This is your login device.

AMITY                   ! This is the DECnet node name of
                        ! your system.

22                      ! This is the baud-rate code for
                        ! your terminal.

15                      ! This is the type code for the
                        ! terminal you are using.
$ @ <EOF>
$
```

As you can see, the appropriate information has been written into the new file.

Notice that in the command file, there are two apostrophes in "today"s," but only one apostrophe appears in the display. When substitution mode is enabled, you must use two apostrophes in any comments so that the text that contains one apostrophe shows up correctly. When you use only one apostrophe, Indirect assumes the text following the apostrophe to be a string symbol. See Chapter 2 for more information.

Notice also in INFO.DAT that a code of 22_8 for <TISPED> means that the baud rate of the terminal is 9600. A code of 15_8 for <TITYPE> means that the terminal is a VT100. See Chapter 2 for complete lists of the codes for the baud rates and terminal types.

- You can also use Indirect directly from the terminal without running a command file. The following command line lets you work with Indirect interactively:

```
>@TI: [RET]
AT.>
```

When Indirect responds with AT.> (the task-name prompt), you can enter Indirect command lines, invoke command files, or display the values of special symbols. To display a symbol, use the .ENABLE SUBSTITUTION directive, then request the symbol in the following format:

```
AT.>;'<symbol>'
```

For example, if you type

```
AT.>.enable substitution [RET]
AT.>;'<time>' [RET]
```

Indirect displays

```
>;15:57:56
AT.>
```

The semicolon in front of the symbol indicates that Indirect should display the time on the terminal, but the CLI should not try to execute it as one of its commands.

To exit from Indirect, press CTRL/Z:

```
AT.> [CTRL/Z]
>@ <EOF>
```

Chapter 2

The Indirect Command Processor (Reference Section)

This chapter describes indirect command files and the Indirect Command Processor (Indirect). Also included are descriptions of the processor directives and symbols that control the execution of Indirect.

Note that parameter defaults and ranges for the Indirect directives are specified in the build file for the Indirect task. These values are installation-specific and can be changed. The values for initial command line and default directory processing, which are discussed in Section 2.2, are also installation-specific and can be changed. If you need more information, ask your system manager.

2.1 Indirect Command Files

Indirect command files can be used to execute many different things—from simple tasks to complex system-control and programming functions.

There are two kinds of indirect command files: indirect task command files and indirect command line interpreter (CLI) command files. The following sections describe these files.

2.1.1 Indirect Task Command Files

An indirect task command file is a text file containing a list of task-specific command lines. Rather than typing and retyping a commonly used sequence of commands and responding to the task's prompts, you can type the sequence once, store it in a file, and direct the task to read the file for its commands. Tasks respond to command lines contained in an indirect command file as if they were entered directly from the terminal. Most system-supplied tasks on RSX-11M-PLUS operating systems, such as MACRO-11 or the Task Builder, accept indirect task command files.

To initiate indirect task command files, replace the command line for a task with a file specification for the command file, preceded by an at sign (@). The task requesting input then accesses the specified file and starts to read and respond to the command lines contained within it. For example, to initiate a file of MACRO-11 command lines from MCR, type the following:

```
>MAC @INPUT.CMD [RET]
```

The MACRO-11 Relocatable Assembler accesses the file INPUT.CMD and executes the command lines contained in it.

The default file type for indirect task command files is CMD. Thus, the command line in the previous example could also be input as follows:

```
>MAC @INPUT [RET]
```

Some tasks allow nested command files (one file invokes another). See the appropriate task documentation for the maximum nesting depth allowed.

Note that indirect task command files can contain valid task-specific command lines only. The Indirect directives (which are described later in this chapter) cannot be used for such command files.

2.1.2 Indirect CLI Command Files

An indirect CLI command file is a text file containing CLI command lines and special directives that allow you to control command file processing. The Indirect Command Processor (which usually runs under the task name AT.) reads the indirect command file, interprets the directives, and passes the CLI commands to the CLI. The CLI can be MCR, DCL, or a user-written CLI.

For example, an indirect command file could contain the following command lines:

```
.ENABLE SUBSTITUTION
.ASKS COMMAN Enter command name
HELP 'COMMAN'
```

With this file, Indirect processes the first two command lines and the CLI executes the HELP command line.

To initiate an indirect command file, type in the file specification preceded by an at sign (@). For example:

```
$ @COMMANDS.CMD [RET]
```

The default file type for indirect CLI command files is also CMD, or CMF for command files that have been pre-processed (see Chapter 3). Thus, the command line in the previous example could also be input as follows:

```
$ @COMMANDS [RET]
```

The name of the indirect command file can also be a logical name assignment that translates into a valid Files Control Services (FCS) file specification. For example,if you have assigned the logical name TEST to the string DU1:[USER]COMMANDS.CMD, the command @TEST invokes the file COMMANDS.CMD in the directory [USER] on the disk DU1.

If the catchall task (TDX) is installed on your system as ...CA., you can give the command file a 3-character name and execute the file without using the at sign. For example:

```
$ ABC [RET]
```

Indirect searches for a command file called ABC.CMF or ABC.CMD.

Indirect CLI command files can also be nested. The maximum level of nesting depends on the version of the Indirect task you are using. Ask your system manager about the version or look at the build file for the task. To illustrate, a maximum nesting level of four means that you can run one command file, which can run another file, which can run a third file, which can run a fourth file.

For example, the following command file executes a DCL command line and then invokes another command file (COOKIE.CMD). When Indirect is finished with COOKIE.CMD, it returns to the first file, which executes more DCL commands.

```
SET TERMINAL/LOWER/SCOPE/WIDTH:80
@COOKIE
SHOW DEVICES
SHOW USERS
SHOW TIME
```

For CLI commands and for questions displayed by the .ASKx directives, the first character displayed is the current CLI prompt. For MCR, the default prompt is the right angle bracket (>). For DCL, the default prompt is a dollar sign followed by a space ($).

The Indirect directives described in Section 2.6 can be used in indirect CLI command files. All further references in this chapter to indirect command files apply to indirect CLI command files.

2.2 The Indirect Command Processor

When processing an indirect command file, Indirect first reads the command file and interprets each command line either as a command to be passed directly to the current CLI or as a request for action by Indirect. The directives for Indirect are distinguished by a period (.) as their beginning character.

The Indirect directives allow you to perform the following functions:

- Define and assign values to logical, numeric, and string symbols (see Section 2.4 for more information on symbols)
- Substitute a symbol's value into any line of the command file
- Perform arithmetic symbol operations
- Manipulate strings
- Display text on the user's terminal
- Ask questions of a user
- Control the sequence of execution of a command file
- Call subroutines
- Detect error conditions
- Test symbols and conditions
- Create and access data files
- Parse commands and data
- Enable or disable any of several operating modes

- Control time-based and parallel task execution
- Expand logical name assignments

These functions are described throughout Section 2.6.

Two directives, .BEGIN and .END, allow you to block-structure the command file and create Begin-End blocks. Modular, block-structured command files are easier to debug and maintain. More importantly, Begin-End blocks isolate local symbol definitions as well as labels and thus conserve symbol table space.

When you define a symbol, Indirect creates an entry for the definition in an internal symbol table. Normally, symbol table entries retain their definitions under the following conditions:

- If defined locally, throughout the execution of the command file.
- If defined globally, throughout the execution of all levels of nested command files (a dollar sign ($) at the beginning of the symbol indicates a global symbol).

When defined within a Begin-End block, however, local symbols retain their definitions only throughout the execution of the commands within that block. The symbols are erased from the symbol table when Indirect encounters the .END directive at the end of the block.

One Indirect directive, .ENABLE GLOBAL (see Section 2.6.12), and a switch, /LO (see Section 2.5), allow the definition of some symbols as global to all file levels. If symbols are *not* global, each time Indirect enters a deeper level, it masks out of the symbol table all symbols defined by the previous level so that only the symbols defined in the current level are available for use by that level. When control returns to a previous level, the symbols defined in that level become available once again and the ones from the lower level or levels are lost.

When Indirect reaches the end of the highest-level indirect command file, it displays the message

```
@ <EOF>
```

and then exits. (The message is not displayed if the .DISABLE DISPLAY directive is in effect. See Sections 2.6.11 and 2.6.12 for more information.)

Indirect displays on the requesting terminal every CLI command line as it is executed. However, if Indirect is activated by @filename/NOCLI, the CLI command lines are displayed but not executed. (See Section 2.5 for information on the /[NO]CLI switch.)

A command file can also include comments. Comments can be placed at different locations in the file and require different preceding characters depending on how you want Indirect and the CLI to treat them. Following are the three formats for comments:

;comment	Comments at beginning of line to be displayed by the CLI
!comment	Comments after the start of a CLI command line
.;comment	Comments that will not be displayed

Indirect attaches the terminal while processing contiguous comment lines that begin with a semicolon. This allows you to press CTRL/O and suppress a lengthy comment. Output is resumed by pressing another CTRL/O or is resumed at the next CLI command line or Indirect directive statement in the command file.

Note that command and comment lines are not displayed if .ENABLE QUIET is in effect (see Section 2.6.12). When Indirect processes a .ENABLE QUIET statement, it forces a detachment (if detach mode is enabled, which is the default) because it no longer needs the terminal for processing. Once quiet mode has been established, no attempts are made to reattach the terminal.

Any CLI command line issued by Indirect also causes an unconditional detachment. This action prevents a task, which may need the terminal, from suspending activity because the terminal is attached by Indirect.

A .DISABLE QUIET statement establishes terminal I/O but does not attempt to detach the terminal. See Section 2.6.12 for more information.

References to task names in an indirect command file follow the rules used for MCR and DCL. If the task was started as an external CLI task (for example, MAC, PIP, DMO), it may be referenced by its 3-character name (xxx). Thus, such directives as .IFINS, .IFACT, and .WAIT need only specify the 3-character task name; Indirect can then find the correct task. However, you can always refer to a specific task by using its full 6-character name (...xxx or xxxnnn).

If you do not specify a file name in the initial command line, Indirect can construct the name of a default file to be opened. The default file is named INDINIxxx.CMD, where xxx is null or the 3-character task name under which Indirect is installed. Note, however, that this facility is usually disabled. To enable it, the value in the build file for the Indirect task must be changed.

If a specified command file cannot be found in the current directory, Indirect can also search for the file in another directory. However, to enable this facility, the value D$CUIC in the build file for the Indirect task must be changed to be a value other than zero. If the new value is 1, Indirect searches for the file in LB:[libuic]. If the new value is greater than 377_8, Indirect considers it to be the octal equivalent of the directory (on LB) to be searched. For example, if you issue the command @ABC.CMD but Indirect cannot find the file in the current directory, if the value of D$CUIC is set to 141454_8, Indirect searches directory [303,54] for the file.

2.3 Summary of Indirect Directives

The Indirect directives described later in this chapter are listed here by category. A detailed description of each directive is given in alphabetical order in Section 2.6.

Category	Function
Label Definition	
.label:	Assigns a name to a line in the command file so that the line may be referenced elsewhere within the file by a .GOTO or .GOSUB directive.
Symbol Definition	
.ASK	Prompts for user input to define or redefine a logical symbol and assign the symbol a true or false value.
.ASKN	Prompts for user input to define or redefine a numeric symbol and assign the symbol a numeric value.

Category	Function
.ASKS	Prompts for user input to define or redefine a string symbol and assign the symbol a character string value.
.ERASE	Deletes all local or global symbol definitions or a single local or global symbol definition.
.SETT .SETF	Defines or redefines a logical symbol and assigns the symbol a true or false value.
.SETN	Defines or redefines a numeric symbol and assigns the symbol a numeric value.
.SETD .SETO	Redefines the radix of a numeric symbol.
.SETL	Defines or redefines a logical symbol and assigns the symbol a true or false value.
.SETS	Defines or redefines a string symbol and assigns the symbol a character string value.
.TRANSLATE	Expands a logical name translation into the special symbol <EXSTRI> .
File Access	
.CHAIN	Closes the current indirect command file and begins executing commands from another file.
.CLOSE	Closes a user data file.
.DATA	Specifies a single line of data to be output to a data file that is already open.
.OPEN	Creates and opens an output data file. (If the file exists, creates a new version and opens it.)
.OPENA	Opens an existing data file and appends subsequent text to it (but does not create a new version). Defaults to .OPEN if the file does not exist.
.OPENR	Opens a data file for reading with the .READ directive.
.PARSE	Parses (divides) strings into substrings.
.READ	Reads a line from a file into a specified string variable.
Logical Control	
.BEGIN	Marks the beginning of a Begin-End block.
.END	Marks the end of a Begin-End block.

Category	Function
.EXIT	Terminates processing of either Indirect or the current command file, returns control to the invoking terminal or to the previous Indirect file level, and optionally sets the value for the special symbol <EXSTAT> .
.GOSUB	Calls a subroutine within the command file.
.GOTO	Branches to a label within the command file.
/	Defines logical end-of-file. Terminates file processing and exits. This directive is equivalent to the .STOP directive. It is the only directive that does not begin with a period and does not consist of alphabetic characters.
.ONERR	Branches to a label upon detecting a specific Indirect error condition.
.RETURN	Effects an exit from a subroutine and returns to the line immediately following the subroutine call.
.STOP	Terminates indirect command file processing and optionally sets Indirect exit status. This directive is equivalent to the logical end-of-file (/) directive.
Logical Tests	
.IF	Determines whether or not a symbol satisfies a condition.
.IFACT .IFNACT	Determines whether or not a task is active.
.IFDF .IFNDF	Determines whether or not a symbol is defined.
.IFENABLED .IFDISABLED	Tests the .ENABLE or .DISABLE options.
.IFINS .IFNINS	Determines whether or not a task is installed in the system.
.IFLOA .IFNLOA	Determines whether or not a device driver is loaded.
.IFT .IFF	Determines whether a logical symbol is true or false.
.TEST	Tests the length of a string symbol or locates a substring.
.TESTDEVICE	Returns information about a device in the system.
.TESTFILE	Determines whether a specified file exists and determines the physical device associated with a logical device name (performs device translation).
.TESTPARTITION	Returns information about a memory partition in the system.

Category	Function
Enable or Disable an Operating Mode	
.ENABLE .DISABLE	Enables or disables control of the following modes: Substitution (SUBSTITUTION) Timeout parameter (TIMEOUT) Lowercase-character processing (LOWERCASE) Terminal attachment (ATTACH, DETACH) Output of data to data files (DATA) File deletion (DELETE) Global symbols (GLOBAL) Symbol radix (DECIMAL) Command line echo (QUIET) Command display (TRACE) Field display (DISPLAY) Passing commands to CLI (CLI) Input truncation error suppression (TRUNCATE) Escape recognition (ESCAPE) Escape-sequence processing (ESCAPE-SEQ) Control-Z recognition (CONTROL-Z) Numeric overflow (OVERFLOW)
Increment or Decrement Numeric Symbols	
.DEC	Decrements the value of a numeric symbol by 1.
.INC	Increments the value of a numeric symbol by 1.
Execution Control	
.DELAY	Delays the execution of an indirect command file for a specified period of time.
.PAUSE	Temporarily suspends the execution of an indirect command file to allow user action.
.WAIT	Waits for a specified task to complete execution and sets the special symbol <EXSTAT> with the completed task's exit status.
.XQT	Initiates a task, passes a command line to it, and continues Indirect processing without waiting for the task to complete.
Special Purpose	
.FORM	Provides access to the FMS-11/RSX form driver. The directive allows FMS commands to be passed to FMS.

2.4 Symbols

Indirect allows you to define symbols. These symbols can then be tested or compared to control flow through the indirect command file. Their values may also be inserted into CLI commands, data records for data files, or comments to be displayed on the terminal.

Symbol names are ASCII strings from one to six characters in length. They must start with a letter (A to Z) or a dollar sign ($). The remaining characters must be alphanumeric or a dollar sign.

There are three symbol types:

- Logical
- Numeric
- String

A logical symbol has a value of either true or false.

A numeric symbol can have a numeric value in the range of 0 to 177777_8 ($65{,}535_{10}$). The symbol can be defined to have either a decimal or octal radix. The radix is relevant only when the symbol is substituted (see Section 2.4.2).

A string symbol has as its value a string of ASCII characters. The string can be 0 to 132_{10} characters in length.

A symbol's type (logical, numeric, or string) is defined by the first directive that assigns a value to the symbol. Assignment directives can assign

- A true or false value to define a logical symbol (defined by .ASK, .SETL, .SETT, or .SETF)
- An octal or decimal number to define a numeric symbol (defined by .ASKN or .SETN)
- A character string to define a string symbol (defined by .ASKS, .READ, or .SETS)

2.4.1 Special Symbols

Indirect defines certain special symbols automatically. These symbols are dependent on specific system characteristics and the replies to queries given during command file execution. Special symbols can be compared, tested, or substituted, and may be one of three types: logical, numeric, or string. All special symbols have a common format: angle brackets (< >) enclose the special symbol name.

Sections 2.4.1.1 to 2.4.4 give brief descriptions of the special logical, numeric, and string symbols, and discuss the use of numeric, string, and logical symbols and expressions. Section 2.4.5 explains reserved symbols, and Sections 2.4.6 and 2.4.6.1 discuss symbol-value substitution.

2.4.1.1 Special Logical Symbols

The special logical symbols are assigned a true or false value based on the following conditions:

Symbol	Value
<ALPHAN>	Set to true if last string entered in response to a .ASKS directive or tested with a .TEST directive contains only alphanumeric characters. An empty string also sets <ALPHAN> to true.
<ALTMOD>	Set to true if last question was answered with an ALTMODE or ESCAPE. Otherwise, <ALTMOD> is set to false.
<BASLIN>	Set to true if the current operating system is a baseline configuration. (This option is used during system-generation to determine what resources are available for the system-generation process.) Otherwise, <BASLIN> is set to false.
<DEFAUL>	Set to true if the answer to the last query was defaulted (the RETURN key was pressed once) or a timeout occurred.
<EOF>	Set to true if the last .READ or .ASKx directive resulted in reading past the end of the file. Otherwise, <EOF> is set to false. <EOF> is also set to true if the last .TRANSLATE directive resulted in a final logical translation assignment.
<ERSEEN>	Set to true if any of the following conditions are true (<ERRNUM> , <EXSTAT> , and <FILERR> are described in Section 2.4.1.2): • <FILERR> is less than 0 (that is, if a negative error code was returned). • An exit status (<EXSTAT>) value more serious than <WARNING> was returned. • <EOF> is set to true. • <ERRNUM> is not 0. • You used the command line .SETT <ERSEEN> . The command line .SETF <ERSEEN> sets the following conditions: • <FILERR> is set to 0. • <EXSTAT> is set to 0. • <EOF> is set to false. • <ERRNUM> is set to 0.
<ESCAPE>	Set to true if last question was answered with an ALTMODE or ESCAPE. Otherwise, <ESCAPE> is set to false. <ESCAPE> is a read-only symbol.
<FALSE>	Logical constant used for comparisons with the .IF directive or as a default for the .ASK directive.

Symbol	Value
<IAS>	Set to true if the current operating system is IAS. Always false on RSX-11M-PLUS systems.
<LOCAL>	Set to true if the terminal from which Indirect is executing (TI) is a local terminal. If the terminal is remote, <LOCAL> is set to false.
<MAPPED>	Set to true if the system on which Indirect is running is mapped; set to false if the system is unmapped. Always true on RSX-11M-PLUS systems.
<NUMBER>	Set to true if the last string entered in response to a .ASKS directive or tested with a .TEST directive contains only numeric characters. An empty string also sets <NUMBER> to true.
<OCTAL>	Set to true if the answer to the last .ASKN directive or the radix of the numeric symbol tested in the last .TEST directive is octal, or if the last string tested with a .TEST directive contained all numeric characters in the range 0 to 7.
<PRIVIL>	Set to true if the current user is privileged. Its value is determined from the flag contained in the terminal database. The symbol is set when Indirect is started and remains unchanged during execution. The next time Indirect is started, <PRIVIL> is reset if a command to change the user's privilege (for example, DCL SET TERM/NOPRIV) was issued during the previous execution.
<RAD50>	Set to true if the last string entered in response to a .ASKS directive or tested with a .TEST directive contains only Radix-50 characters. Radix-50 characters are the uppercase alphanumeric characters plus period (.) and dollar sign ($). A blank is not a Radix-50 character in this context. An empty string also sets <RAD50> to true.
<RSX11D>	Set to true if the current operating system is RSX-11D. Always false on RSX-11M-PLUS systems.
<TIMOUT>	Set to true if timeout mode is enabled and the last .ASKx directive timed out waiting for a user response.
<TRUE>	Logical constant used for comparisons with the .IF directive or as a default for the .ASK directive.

2.4.1.2 Special Numeric Symbols

The special numeric symbols are assigned the following values:

Symbol	Value
<ERRCTL>	Controls the way in which Indirect processes errors. The symbol is treated as an 8-bit mask. For each class of error that a user's .ONERR target routine processes (see Section 2.6.22), the appropriate bit is set in the mask. If the bit is cleared, Indirect exits after printing the error information. If the eighth bit, which is the sign bit or 200_8, is set, Indirect does not print any information about the error. The initial default value for <ERRCTL> is 1, which implies that only class 1 errors can be handled with a .ONERR address and that error messages will be printed. To cover class 1 and 2 errors, the value for <ERRCTL> must be 3. **Note** If you attempt to trap errors other than default class 1, processing cannot continue in most cases. The error service routine is limited to returning a fatal error message and executing the .EXIT directive. The internal state of Indirect is indeterminate in all but class 1 error cases. If you receive an error that is not class 1, clean up what you are doing as much as possible and exit from Indirect. See Appendix A for a list of the error messages and their assigned class values.
<ERRNUM>	Assigned the class number of an error that Indirect has finished processing. This value can be used for processing specific error types with a .ONERR routine. See Appendix A for a list of the error messages and their assigned class values.
<ERRSEV>	Assigned the error severity mask associated with the error that Indirect has finished processing. This bit mask corresponds to the bit mask <ERRCTL> used to control the processing.
<EXSTAT>	Assigned the value of 0, 1, 2, 4, or 17, depending on the exit status from the last CLI command line executed or from the last ".WAIT taskname" directive, where taskname was activated by the .XQT directive. <EXSTAT> is modified at the completion of a synchronous CLI command line or at the completion of a .WAIT directive. The .EXIT directive can also modify <EXSTAT> . The value is returned from a task that has completed if the task exits with status. Otherwise, the value is returned from the CLI. The values 0, 1, 2, 4, and 17 and their corresponding special symbols indicate:

Symbol	Value		
	0	<WARNIN>	Warning
	1	<SUCCES>	Success
	2	<ERROR>	Error
	4	<SEVERE>	Severe error
	17	<NOSTAT>	The task could not return exit status.

<FILERR> Assigned the FCS-11 (I/O error or driver) or directive (DSW) status code resulting from a .TESTFILE, .OPENx, or .READ directive operation. <FILERR> contains the contents of offset F.ERR and F.ERR+1 from the File Descriptor Block (FDB) associated with the file. If F.ERR+1 (the high byte of the word) contains 0, F.ERR (the low byte of the word) contains an I/O error code. If F.ERR+1 contains –1, F.ERR contains a directive status code.

The following lists give the codes (in octal words) and their meanings.

I/O Error Codes (F.ERR+1, the high byte, contains 0):

Error Number		Meaning
Decimal	Octal	
–1	000377	Bad parameters
–2	000376	Invalid function code
–3	000375	Device not ready
–4	000374	Parity error on device
–5	000373	Hardware option not present
–6	000372	Illegal user buffer
–7	000371	Device not attached
–8	000370	Device already attached
–9	000367	Device not attachable
–10	000366	End-of-file detected
–11	000365	End-of-volume detected
–12	000364	Write attempted to locked unit
–13	000363	Data overrun

Symbol	Value	
	Error Number	**Meaning**
-14	000362	Send/receive failure
-15	000361	Request terminated
-16	000360	Privilege violation
-17	000357	Shareable resource in use
-18	000356	Illegal overlay request
-19	000355	Odd byte count (or virtual address)
-20	000354	Logical block number too large
-21	000353	Invalid UDC module number
-23	000351	Caller's nodes exhausted
-24	000350	Device full
-25	000347	Index file full
-26	000346	No such file
-27	000345	Locked from read/write access
-28	000344	File header full
-29	000343	Accessed for write
-30	000342	File header checksum failure
-31	000341	Attribute control list format error
-32	000340	File processor device read error
-33	000337	File processor device write error
-34	000336	File already accessed on LUN
-35	000335	File ID, file number check
-36	000334	File ID, sequence number check
-37	000333	No file accessed on LUN
-38	000332	File was not properly closed

Symbol	Value	
	Error Number	**Meaning**
-39	000331	No buffer space available for file
-40	000330	Illegal record size
-41	000327	File exceeds space allocated, no blocks
-42	000326	Illegal operation on FDB
-43	000325	Bad record type
-44	000324	Illegal record-access bits set
-45	000323	Illegal record-attribute bits set
-46	000322	Illegal record number—too large
-47	000321	Internal consistency error
-48	000320	Rename—two different devices
-49	000317	Rename—a new file name already in use
-50	000316	Bad directory file
-51	000315	Cannot rename old file system
-52	000314	Bad directory syntax
-53	000313	File already open
-54	000312	Bad file name
-55	000311	Bad device name
-56	000310	Bad block on device
-57	000307	Enter—duplicate entry in directory
-58	000306	Not enough stack space (FCS or FCP)
-59	000305	Fatal hardware error on device
-60	000304	File ID was not specified
-61	000303	Illegal sequential operation
-62	000302	End-of-tape detected
-63	000301	Bad version number
-64	000300	Bad file header

Symbol	Value	
	Error Number	**Meaning**
	-65 000277	Device off line
	-66 000276	Block check, CRC, or framing error
	-67 000275	Device on line
	-68 000274	No such node
	-69 000273	Path lost to partner
	-70 000272	Bad logical buffer
	-71 000271	Too many outstanding messages
	-72 000270	No dynamic space available
	-73 000267	Connection rejected by user
	-74 000266	Connection rejected by network
	-75 000265	File expiration date not reached
	-76 000264	Bad tape format
	-77 000263	Not ANSI "D" format byte count
	-78 000262	No data available
	-79 000261	Task not linked to specified ICS/ICR interrupts
	-80 000260	Specified task not installed (.NST)
	-80 000260	No AST specified in connect (.AST)
	-81 000257	Device off line when offline request was issued
	-82 000256	Invalid escape sequence
	-83 000255	Partial escape sequence
	-84 000254	Allocation failure
	-85 000253	Unlock error
	-86 000252	Write-check failure
	-87 000251	Task not triggered
	-88 000250	Transfer rejected by receiving CPU
	-89 000247	Event flag already specified

Symbol	Value	

Error Number		Meaning
-90	000246	Disk quota exceeded
-91	000245	Inconsistent qualifier usage
-92	000244	Circuit reset during operation
-93	000243	Too many links to task
-94	000242	Not a network task
-95	000241	Timeout on request
-96	000240	Connection rejected
-97	000237	Unknown name
-98	000236	Unable to size device
-99	000235	Media inserted incorrectly
-100	000234	Spindown ignored

Directive Status Codes (F.ERR+1, the high byte, contains -1):

Error Number		Meaning
Decimal	Octal	
-1	177777	Insufficient dynamic storage
-2	177776	Specified task not installed
-3	177775	Partition too small for task
-4	177774	Insufficient dynamic storage for send
-5	177773	Unassigned LUN
-6	177772	Device handler not resident
-7	177771	Task not active
-8	177770	Directive inconsistent with task state
-9	177767	Task already fixed/unfixed
-10	177766	Issuing task not checkpointable
-11	177765	Task is checkpointable
-15	177761	Receive buffer is too small
-16	177760	Privilege violation

Symbol	Value	
	Error Number	**Meaning**
	-17 177757	Resource in use
	-18 177756	No swap space available
	-19 177755	Illegal vector specified
	-20 177754	Invalid table number
	-21 177753	Logical name not found
	-80 177660	Directive issued/not issued from AST
	-81 177657	Illegal mapping specified
	-83 177655	Window has I/O in progress
	-84 177654	Alignment error
	-85 177653	Address window allocation overflow
	-86 177652	Invalid region ID
	-87 177651	Invalid address window ID
	-88 177650	Invalid TI parameter
	-89 177647	Invalid send buffer size (greater than 255)
	-90 177646	LUN locked in use
	-91 177645	Invalid UIC
	-92 177644	Invalid device or unit
	-93 177643	Invalid time parameters
	-94 177642	Partition/region not in system
	-95 177641	Invalid priority (greater than 250)
	-96 177640	Invalid LUN
	-97 177637	Invalid event flag (greater than 64)
	-98 177636	Part of DPB out of user's space
	-99 177635	DIC or DPB size invalid

See the *RSX-11M-PLUS and Micro/RSX I/O Operations Manual* and the *RSX-11M-PLUS and Micro/RSX I/O Drivers Reference Manual* for more information.

<FILER2> Assigned the second error-code word returned by the FMS-related commands. See the description of the .FORM directive (Section 2.6.16) for more information.

Symbol	Value
<FORATT>	Assigned the octal value of the file attributes that were used to open the data files.
<MEMSIZ>	Assigned the value of the current system memory size in K words (K is 1024_{10}).
<REGSEG>	Assigned the number, in octal, of groups of free bytes in the internal string symbol storage region. This symbol can be used to detect severe fragmentation in the region, which occurs because the region is not shuffled after each addition or deletion of a variable. Severe fragmentation may indicate the need to delete variables you no longer need to make more storage space available in the region.
<REGSIZ>	Assigned the number, in octal, of free bytes in the internal string storage region.
<SPACE>	Assigned the number, in octal, of free bytes in the internal symbol table for Indirect. The number does not reflect the amount of space that could be gained by the automatic extension of the Indirect task.
<STRLEN>	Assigned the length, in octal, of the string entered in response to the last .ASKS directive or the string tested by the last .TEST directive. The symbol is also set when a command file is invoked (<STRLEN> contains the octal number of variables used in the command line) and as the result of a .PARSE statement (<STRLEN> contains the octal number of substrings produced by the directive).
<SYMTYP>	Assigned the numeric code for the type of symbol tested with a .TEST directive. The symbol types have the following code numbers: Logical – 0 Numeric – 2 String – 4
<SYSTEM>	Assigned an octal number to represent the operating system on which Indirect is running. For an RSX-11M-PLUS system, the value is 6. For a VAX-11 RSX system, the value is 5.
<SYUNIT>	Assigned the unit number of the user's default device (SY).
<TICLPP>	Assigned the current page length setting for the terminal. When you first invoke Indirect, it attempts to determine the length of the page. If the information is not available, <TICLPP> defaults to 24_{10}.
<TICWID>	Assigned the current page width setting for the terminal. When you first invoke Indirect, it attempts to determine the width of the page. If the information is not available, <TICWID> defaults to 80_{10}.

Symbol	Value
<TISPED>	Assigned the baud rate for transmitting characters from the host system to the terminal. When you first invoke Indirect, it attempts to determine the baud rate. The baud-rate information is useful for determining the quality and quantity of information to be transmitted. The following list gives the octal value that corresponds to the baud rates:

Octal Value	Baud Rate	Octal Value	Baud Rate
1	0	13	1200
2	50	14	1800
3	75	15	2000
4	100	16	2400
5	110	17	3600
6	134	20	4800
7	150	21	7200
10	200	22	9600
11	300	23	EXTA
12	600	24	EXTB

Symbol	Value
<TITYPE>	Assigned the terminal type of the terminal from which Indirect is running. If the terminal type is changed from within an indirect command file, <TITYPE> is set to the latest terminal type. If Indirect cannot determine the terminal type, <TITYPE> is set to zero (0). The following list gives the octal value that corresponds to the terminal types:

Value-Type	Value-Type	Value-Type
0—Unknown	11—VT52	23—LA38
1—ASR33	12—VT55	24—VT101
2—KSR33	13—VT61	25—VT102
3—ASR35	14—LA180S	26—VT105
4—LA30S	15—VT100	27—VT125
5—LA30P	16—LA120	30—VT131
6—LA36	20—LA12	31—VT132
7—VT05	21—LA100	35—PC3xx-series
10—VT50	22—LA34	36—VT2xx-series

See the *RSX-11M-PLUS and Micro/RSX I/O Drivers Reference Manual* (the chapter on the full-duplex terminal driver) for more information.

2.4.1.3 Special String Symbols

The special string symbols are assigned the following string values:

Symbol	Value
<ACCOUN>	Assigned certain accounting information from a user's accounting block (UAB). If Resource Accounting is not running on the system, the fields of <ACCOUN> are null. The information is in the following format (note the trailing comma): username,sessionid,accountnumber,CPU,DIR,QIO,TAS,activetasks, username — The first 14_{10} characters of the user name (as it appears in the system account file) followed by the first initial. sessionid — The 3-letter session-ID code followed by the unique login number. accountnumber — The user's account number as it appears in the system account file. CPU — The number of CPU ticks used since login. DIR — The number of system directives issued since login. QIO — The number of QIO directives issued since login. TAS — The number of tasks run since login. activetasks — The current number of the user's active tasks. The individual fields can be isolated with the .PARSE directive: `.PARSE <ACCOUN> "," NAME SID ACNT CPU DIR QIO TAS ACT JUNK` Note that because double-precision arithmetic is not available in Indirect, the numeric <ACCOUN> parameters cannot be converted to numeric form and manipulated in arithmetic expressions.
<CLI>	Assigned the acronym (three to six letters) of the current command line interpreter (for example, DCL).
<CONFIG>	Contains the parameter defaults specified when the current Indirect task was built. See the module INDCFG in the system procedure library LB:[1,2]INDSYS.CLB on the system disk for more details.
<DATE>	Assigned the current date; format is dd-mmm-yy.

Symbol	Value
<DIRECT>	Contains a user's current default directory string; format is [name]. The contents of <DIRECT> are different depending on the directory mode you are in and the kind of directory you are using. If you are not in named directory mode ("nonamed" mode), <DIRECT> contains a null directory string ([]). If you are in named directory mode and using a named directory, <DIRECT> contains your default directory string in the form [ddddddddd]. If you are in named directory mode and using a numeric directory, <DIRECT> contains your default directory string in the form [gggmmm]. If <DIRECT> contains the null string ([]), use the special symbol <UIC> for the location of your current default directory. If <DIRECT> contains a directory string, use it as the current default directory location.
<EXSTRI>	When Indirect is first initiated, contains build-time information about the Indirect task. The information includes the version number of the task and the time the task was built. Afterwards, it can contain such information as the string results from a more deeply nested indirect command file or the results of a .TESTDEVICE statement. The results are sent to the calling command file. This symbol can be redefined with a .SETS <EXSTRI> xxxx command.
<FILATR>	Contains eight fields of file-attribute information obtained from offsets for the File Descriptor Block (FDB). The information is from the FDB used in the last .OPENx operation and is in the following format (note the trailing comma): rtyp,ratt,rsiz,hibk,efbk,ffby,racc,rctl, The attributes are: F.RTYP Record type (byte, octal). Set as follows to indicate the type of records for the file: 1—fixed-length records (R.FIX) 2—variable-length records (R.VAR) 3—sequenced records (R.SEQ) F.RATT Record attribute (byte, octal). Bits 0 to 3 are set as follows to indicate record attributes: Bit 0 If 1, first byte of record contains a FORTRAN carriage control character (FD.FTN); otherwise, 0. Bit 1 If 1, for a carriage control device, a line feed is to occur before the line is printed and a carriage return is to occur after the line is printed (FD.CR); otherwise, 0. Bit 2 If 1, indicates print file format (FD.PRN); FCS allows this attribute but does not interpret the format word; otherwise, 0. Bit 3 If 1, the records cannot cross block boundaries (FD.BLK); otherwise, 0.

Symbol	Value
F.RSIZ	Record size (word, decimal). Contains the size of fixed-length records or indicates the size of the largest record that currently exists in a file of variable-length records.
F.HIBK	Highest virtual block number allocated (double word with F.EFBK, decimal).
F.EFBK	End-of-file block number (double word with F.HIBK, decimal).
F.FFBY	First free byte in the last block or the maximum block size for magnetic tape (word, octal).
F.RACC	Record access (byte, octal). Bits 0 to 3 define as follows the record access modes: Bit 0 — If 1, READ$/WRITE$ mode (FD.RWM); if 0, GET$/PUT$ mode. Bit 1 — If 1, random access mode (FD.RAN) for GET$/PUT$ record I/O; if 0, sequential access mode. Bit 2 — If 1, locate mode (FD.PLC) for GET$/PUT$ record I/O; if 0, move mode. Bit 3 — If 1, PUT$ operation in sequential mode does not truncate the file (FD.INS); if 0, PUT$ operation in sequential mode truncates the file.
F.RCTL	Device characteristics (byte, octal). Bits 0 to 5 define as follows the characteristics of the device associated with the file: Bit 0 — If 1, record-oriented device (FD.REC); if 0, block-structured device. Bit 1 — If 1, carriage control device (FD.CCL); otherwise, 0. Bit 2 — If 1, teleprinter device (FD.TTY); otherwise, 0. Bit 3 — If 1, directory device (FD.DIR); otherwise, 0. Bit 4 — If 1, single-directory device (FD.SDI; an MFD is used, but no directories are present). Bit 5 — If 1, block-structured device inherently sequential in nature (FD.SQD), such as a magnetic tape. Record-oriented devices are assumed to be sequential in nature, so this bit is not set for them.

If no file is currently open, a fatal error occurs.

Symbol	Value
<FILSPC>	Assigned the specification for the file referred to with the last .OPEN, .OPENA, .OPENR, or .TESTFILE directive operation; otherwise, assigned the expanded file specification of the current command file.
<FMASK>	Contains octal values representing answers to some of the system generation questions. Refer to the module INDSFN in the system procedure library LB:[1,2]INDSYS.CLB on the system disk for an explanation of the values.
<LIBUIC>	Assigned the UIC of the current nonprivileged task library; format is [ggg,mmm], where ggg is the group number of the UIC and mmm is the member number of the UIC (leading zeros are not included).
<LOGDEV>	Assigned the device name and unit number of the user's login account.
<LOGUIC>	Assigned the login UIC of the current user ; format is [ggg,mmm].
<NETNOD>	Assigned the DECnet node name of the system. If the system is not on the DECnet network, <NETNOD> is assigned RSX11.
<NETUIC>	If the system has DECnet, assigned the UIC in which DECnet-related tasks are stored on the system volume; format is [ggg,mmm]. <NETUIC> is used with <SYSUIC> and <LIBUIC> to separate the components of the system.
<NXTSYM>	Used as part of a dump routine in the command library file INDSYS.CLB on LB:[1,2]. This is a DIGITAL-supplied routine and it is highly recommended that you not use it.
<SYDISK>	Assigned the device mnemonic (two letters) of the user's default device (SY); format is dd (for example, DU).
<SYSDEV>	Assigned the physical name of the system disk. The device name is in the form ddn (for example, DU0).
<SYSID>	Assigned the operating system's base-level number.
<SYSUIC>	Assigned the system UIC; format is [ggg,mmm].
<SYTYP>	Contains a string consisting of up to 12 ASCII characters that identifies the system. Only four identifications are valid: RSX-11M, RSX-11M-PLUS, Micro/RSX, and VAX-11 RSX.
<TIME>	Assigned the current time; format is hh:mm:ss.

Symbol	Value
<UIC>	Assigned the current UIC. The contents of <UIC> are different depending on the directory mode you are in and the kind of directory you are using. If you are not in named directory mode ("nonamed" mode), <UIC> contains your default UFD in the form [ggg,mmm]. If you are in named directory mode and using a named directory, <UIC> contains your protection UIC in the form [ggg,mmm]. In this case, there is no default. If you are in named directory mode and using a numeric directory, <UIC> contains the default UFD corresponding to the default directory string in <DIRECT> . The default UFD is in the form [ggg,mmm]. (<UIC> follows the numbered default directory string in named directory mode so that all command files and tasks created previous to RSX-11M-PLUS Version 3.0 will still work from named mode in a numeric directory.) If you need to obtain the protection UIC from within an indirect command file, use one of the following procedures: • If you are a privileged user, use the <UIC> symbol. • If you are a nonprivileged user and your system has Resource Accounting, use the <LOGUIC> symbol. If you are a nonprivileged user but your system does not have Resource Accounting, you do not have any way of obtaining the protection UIC.
<VERSN>	Contains a string consisting of up to four ASCII characters that identifies the version number of the system; format is n.n (for example, 4.0).

2.4.2 Numeric Symbols and Expressions

A numeric symbol is a string of digits representing a value in the range of 0 to 177777_8 (0 to $65{,}535_{10}$), if immediately followed by a period or if decimal mode has been enabled). If an arithmetic operation yields a result outside of this range, or one that crosses the boundaries, a fatal error occurs and the following message is displayed (unless turned off by .ENABLE OVERFLOW):

```
AT. -- Numeric under- or overflow
```

A numeric symbol or constant may be combined with another numeric symbol or constant by a logical or arithmetic operator to form a numeric expression. Arithmetic operators are used to add (+), subtract (-), multiply (*), and divide (/). Logical operators are the inclusive OR (!), logical AND (&), and NOT (#). Embedded spaces and tabs are not permitted in front of operators. If a space precedes an operator, particularly the plus sign (+), the operator will not function correctly.

Numeric expressions are evaluated from left to right unless parentheses are used to form subexpressions, which are evaluated first. For example, the directive statements

```
.SETN N1 2
.SETN N2 3
.SETN N3 N1+N2*4
```

assign numeric symbol N3 the value 24_8, whereas the directive statements

```
.SETN N1 2
.SETN N2 3
.SETN N3 N1+(N2*4)
```

assign numeric symbol N3 the value 16_8.

Numeric expressions are permitted as second operands in numeric .IF and .SETN directives. They are also permitted as range and default arguments in .ASKN and .ASKS directives. The directives .EXIT and .STOP allow numeric expressions to represent exit status.

Indirect associates a radix, either octal or decimal, with each numeric symbol. The radix of a numeric symbol changes each time the symbol is assigned a new value. If you use a numeric expression to assign a new value to a symbol and all operands in the expression are octal, then the symbol is set to octal. If any operand in the expression is decimal, the symbol is set to decimal. For example:

```
.SETN N1 2                  ! N1 is octal
.SETN N2 3.                 ! N2 is decimal
.SETN N3 N1+3               ! N3 is octal
.SETN N3 N1+3.              ! N3 is decimal
.SETN N3 N1+N2              ! N3 is decimal
```

You can also assign a new value to a symbol with the .ASKN directive. See Section 2.6.3 for more information.

The .SETO and .SETD directives allow you to change the radix of a numeric symbol without changing the value of the symbol. For example:

```
.SETN N1 10.                ! N1 = 10 decimal
.SETO N1                    ! N1 = 12 octal
```

See Section 2.6.32 for more information on .SETO and .SETD.

The radix of a numeric symbol does not affect arithmetic operations or comparisons. The radix is important only when substituting a numeric symbol into a string. If the radix of the symbol is octal, the value of the symbol is substituted into the string as an octal number. If the radix is decimal, the value is substituted as a decimal number. For example:

```
.SETN N1 10.                ! N1 = 10 decimal
; N1 = 'N1'                 ! Displayed as ; N1 = 10
.SETO N1                    ! Make N1 octal
; N1 = 'N1'                 ! Displayed as ; N1 = 12
```

If you substitute a numeric symbol into a string and the substituted number is decimal, a period (.) following the symbol name causes a trailing period to be included in the string (following the substituted number). For example:

```
.SETN N1 10.                ! N1 = decimal
; N1 = 'N1'                 ! Displayed as ; N1 = 10
; N1 = 'N1.'                ! Displayed as ; N1 = 10.
.SETO N1                    ! Make N1 octal
; N1 + 'N1.'                ! Displayed as ; N1 = 12
```

You can also force a numeric symbol to be substituted as an octal or decimal number by using a substitution format control string. For example:

```
.SETN N1 10.                ! N1 = 10 decimal
; N1 = 'N1%D'               ! Displayed as ; N1 = 10
; N1 = 'N1%O'               ! Displayed as ; N1 = 12
```

Octal is the default radix for symbols substituted using format control strings.

See Section 2.4.5.1 for more information.

2.4.3 String Symbols, Substrings, and Expressions

A string constant is a string of any printable characters enclosed by quotation marks (") or number signs (#). When you begin a string with one of these delimiters, you must end it with the same delimiter. Using number signs is helpful when you want to include quotation marks in the string. Empty strings are also permitted. The number of characters cannot exceed 132_{10}. For example:

```
"ABCDEF"
#HITHERE#
#HI"THERE"#
   ""
   ##
```

String symbols may have the value of any string constant. The value is assigned by a .SETS or .ASKS directive. For example, the directive statements

```
.SETS   S1   "ABCDEF"
.SETS   S2    S1
```

assign string symbol S2 the value of string symbol S1 (that is, ABCDEF).

A substring facilitates the extraction of a segment from the value of a string symbol. You can use substrings only in second operands of .SETS, .IF, and .TEST directives (Format 2). For example, the directive statements

```
.SETS   S1   "ABCDEF"
.SETS  S2    S1[1:3]
```

assign string symbol S2 the value of string symbol S1 beginning at character one and ending at character three (that is, ABC).

You can also use the syntax [n:*] to extract the characters from position n to the end of the string. For example, the directive statements

```
.SETS  S1  "ABCDEF"
.SETS  S2  S1[3:*]
```

assign string symbol S2 the value CDEF.

You can combine a string constant, symbol, or substring with another string constant, symbol, or substring by the string concatenation operator (+) to form a string expression.

String expressions are permitted as second operands in .SETS and .IF directives where the first operand is a string symbol. For example, the directive statements

```
.SETS   S1   "A"
.SETS   S2   "CDEF"
.SETS   S3   S1+"B"+S2[1:3]
```

assign string symbol S3 the value of the concatenation of string symbol S1, string constant "B," and the first three characters of string symbol S2 (that is, ABCDE).

2.4.4 Logical Symbols and Expressions

A logical symbol is a variable that has a value of true or false. A logical constant is one of the following special symbols:

- <TRUE>
- <FALSE>

A logical symbol or constant may be combined with another logical symbol or constant by a logical operator to form a logical expression. Logical operators are the inclusive OR (!), logical AND (&), and NOT (#). Embedded spaces and tabs are not permitted in front of operators.

Logical expressions are evaluated from left to right unless parentheses are used to form subexpressions, which are evaluated first. For example, the directive statement

```
.SETL TEST A!(B&C)
```

sets the logical symbol TEST to true if A is true or if both B and C are true.

Logical symbols can be directly assigned true or false values with the .SETT and .SETF directives. For example, the directive statement

```
.SETT S1
```

sets the symbol S1 to true.

The .SETL directive uses logical operators to evaluate an expression and then sets a logical symbol to true or false. For example, the directive statement

```
.SETL SAMPLE EXA&EXB
```

sets the logical symbol SAMPLE to true if both EXA and EXB are true.

By using the .ASK directive, logical symbols can also be set to true or false, depending on user input. The .ASK directive displays a question on the terminal, waits for a reply, and then sets a specified logical symbol to true or false, depending on the reply. For example:

```
.ASK DISPLY Do you want to display the file?
```

displays the following question on the terminal:

```
* Do you want to display the file? [Y/N]
```

The symbol DISPLY will be set to true or false after you type Y or N or press the RETURN key or the ESCAPE key (if escape recognition is enabled). See Section 2.6.2 for more information.

Logical operators used in an arithmetic expression affect the specified logical operation on the numeric operand or operands on a bit-by-bit basis. For example:

```
.SETN N1 146314          ! Binary pattern 1100110011001100
.SETN N2 125252          ! Binary pattern 1010101010101010
.SETN N3 N1!N2           ! Inclusive OR operator
; N3 = 'N3'              ! Displayed as : N3 = 167356, which is
                         ! binary pattern 1110111011101110
.SETN N4 #N1             ! NOT operator
; N4 = 'N4'              ! Displayed as ; N4 = 31463, which is
                         ! binary pattern 0011001100110011
```

2.4.5 Reserved Symbols

Parameters for a command file can be passed to Indirect for processing. (This is not true for a .CHAIN command line, however.) The parameters are stored in the following reserved local symbols:

P0, P1, P2, P3, P4, P5, P6, P7, P8, P9, COMMAN

The symbol COMMAN contains everything in the issuing command line, including the specification for the command file.

The symbols P0 to P9 contain individual elements of the command line. The elements are delimited by a single space or tab character in between each one. Two delimiting characters between elements represent a null parameter. (See the description of the .PARSE directive in Section 2.6.26 for an example of this behavior.)

With the .GOSUB directive (see Section 2.6.17), any parameters to the right of the label and to the left of a comment are transferred to the symbol COMMAN. The value of COMMAN can then be parsed to obtain formal call parameters.

2.4.6 Symbol Value Substitution

Substitution can occur in any line. Indirect uses the values assigned to logical, numeric, string, or special symbols by replacing a normal parameter (for example, a device unit) with the symbol name enclosed in apostrophes (for example, 'DEVICE'). When a previous directive has enabled substitution mode (.ENABLE SUBSTITUTION), Indirect replaces the symbol name enclosed in apostrophes with the value assigned to the symbol.

When Indirect encounters an apostrophe, it treats the subsequent text, up to a second apostrophe, as a symbol name. Indirect then searches the table of symbols for the corresponding symbol and substitutes the value of the symbol in place of the symbol name and surrounding delimiters in the command line.

The first three lines in the following example appear in an indirect command file. When Indirect executes these lines, it displays the last two lines at the entering terminal.

```
.ENABLE SUBSTITUTION
.ASKS DEVICE Device to mount?
MOUNT 'DEVICE'

>* Device to mount? [S]: DU1: RET
>MOUNT DU1:
```

DU1: was entered in response to the displayed question. This reply assigned the string value DU1: to string symbol DEVICE. Then, when Indirect read

```
MOUNT 'DEVICE'
```

it substituted for 'DEVICE' the value assigned to DEVICE (that is, DU1:). If substitution mode had not been enabled, Indirect would simply have passed the line to the CLI as it appeared in the command file (that is, MOUNT 'DEVICE').

To include an apostrophe as text within a command line rather than as the start of a symbol, you must replace the single apostrophe with two contiguous apostrophes (''). If substitution mode is enabled, Indirect displays the command file line

```
;DON''T PANIC

   as

;DON'T PANIC
```

2.4.6.1 Substitution Format Control

The conversion of numeric values to strings and the placement of string and logical values in a substitution operation can be controlled with a format control string. The control string is in the following form:

'symbol%controlstring'

The control string begins with the percent sign (%) and ends with the second of the two delimiters (apostrophes) that denote the substitution operation. The control string consists of one or more of the following characters:

C Compress leading, embedded, and trailing blanks, and remove embedded nulls (leave one space between characters, but strip all leading and trailing spaces).

D Force the conversion of a numeric symbol to decimal.

O Force the conversion of a numeric symbol to octal.

S Perform signed conversion for a numeric symbol.

M Perform magnitude conversion for a numeric symbol.

Z Return leading zeros for a positive numeric value.

Rn Right-justify the resulting string, truncating to 'n' decimal characters if necessary.

Ln Left-justify the resulting string, truncating to 'n' decimal characters if necessary.

X Convert the variable to Radix-50 characters.

V If the symbol being substituted is numeric, convert the low byte to its equivalent ASCII character and substitute it.

If the symbol being substituted is a string, convert the first character to its octal representation and substitute it.

The default radix for numeric symbols is octal, even if the symbol had been previously declared decimal with a .SETD directive. If the results of the substitution are to be decimal, the D character must be included in the control string.

As an example, the following command file shows various control strings being used and the results of using the control strings:

```
.ENABLE SUBSTITUTION
.SETS STRING "  A B  CD  "
; STRING = 'STRING%C'     ! Compress spaces

.SETS STRING "ABCD"
; STRING = 'STRING%R5'    ! Right-justify string,
;                         ! truncating to 5 characters
; STRING = 'STRING%3'     ! Right-justify string,
;                         ! truncating to 3 characters

.SETN NUMBER 10.
; NUMBER = 'NUMBER%D'     ! Convert numeric symbol to decimal
; NUMBER = 'NUMBER%O'     ! Convert numeric symbol to octal

; NUMBER = 'NUMBER%ZO'    ! Return leading zeros for positive
;                         ! numeric value, convert to octal

; NUMBER = 'NUMBER%ZOR4'  ! Return leading zeros; convert to
;                         ! octal; right-justify,
;                         ! truncating to 4 characters
```

When the command file is executed, Indirect displays the text as follows:

```
>@CONTROL RET
>; STRING = A B CD   ! Compress spaces
>; STRING =  ABCD    ! Right-justify string,
>;                   ! truncating to 5 characters

>; STRING = ABC      ! Right-justify string,
>;                   ! truncating to 3 characters

>; NUMBER = 10       ! Convert numeric symbol to decimal
```

```
>; NUMBER = 12       ! Convert numeric symbol to octal

>; NUMBER = 000012   ! Return leading zeros for positive
>;                   ! numeric value, convert to octal

>; NUMBER = 0012     ! Return leading zeros; convert to
>;                   ! octal; right-justify,
>;                   ! truncating to 4 characters
```

Indirect does not perform a consistency check on the control string. If you specify conflicting format characters, Indirect uses the last one specified.

2.5 Switches

Indirect accepts the following switches: /TR, /CLI, /MC, /LB, /LO, and /DE. Descriptions of the switches are given here.

Switch	Function
/[NO]TR	Displays a trace of the indirect command file on the terminal from which the file is being executed. This function is useful for debugging an indirect command file. Each command line, including Indirect directive statements, is displayed. As each command line is processed, a number representing the nesting depth of the command file is displayed, followed by an exclamation point and the command line. If the command line causes some action to occur, the next displayed line indicates the action; usually, this line consists of the CLI commands issued as a result of the previous directive. The default is /NOTR.
/[NO]CLI	Passes commands not processed by Indirect to your command line interpreter. This switch is synonymous with /[NO]MC. The default is /CLI.
/[NO]MC	Passes commands not processed by Indirect to your command line interpreter This switch is synonymous with /[NO]CLI. The default is /MC.
/LB	Indicates that the specified file is a universal library of command procedures and that the specified module is the procedure to be executed. When command procedures, which are indirect command files, are inserted into a universal library with the LBR /IN command (from MCR) or the DCL LIBRARY/INSERT command, you can then reference them with /LB:module. Command libraries are built by creating a universal library and inserting command files into it. You can then reference the procedures in the library with the following command line: @commandlibrary/LB:module The default file type for a command library is CLB. If you do not specify a module (@commandlibrary/LB), Indirect attempts to locate a module called .MAIN..

Switch	Function

If you do not specify a library name (@/LB:module), the following actions occur:

- If the command is issued from the terminal or from a file that is not in the library, Indirect ignores the /LB switch and treats the command line as though you had used @module.CMD or @module.CMF. Note that if the command is issued from a command file, the default device and directory of the specified module are the same as those for the current file, not necessarily the same as those for the terminal.
- If the command is issued from within a library, the specified module is searched for in the current library.

These default actions for an unspecified library allow a collection of procedures to be developed in a given directory with the @/LB:module or .CHAIN /LB:module commands. When the procedures are then placed in a library, no source changes are required.

Example

The command file PARAM.CMD contains parameter definitions for the .SETN directive and the command file SYSPRC.CMD contains system-specific procedures. The following DCL command lines create the command library and enter the command files into it:

```
$ LIBRARY/CREATE/UNIVERSAL:CMD SYSTART.CLB
$ LIBRARY/INSERT SYSTART.CLB PARAM,SYSPRC
```

You can then use the following command lines to reference the command library modules:

```
$ @SYSTART/LB:PARAM      !Define global symbols
$ @SYSTART/LB:SYSPRC     !Run init procedure
```

DIGITAL supplies a library of command procedures on the system disk. The library is LB:[1,2]INDSYS.CLB and it contains the following procedures:

INDCFG Displays the current build parameters for the running Indirect task.

INDDMP Dumps to the terminal the contents of the Indirect symbol table.

INDPRF A sample procedure to fully parse filename strings.

INDSFN Returns system-configuration information.

INDVFY Displays the values of all of the special symbols.

QIOERR Returns a string expansion of the <FILERR> error codes.

.INDEX Displays an index of the procedures in the library.

Switch	Function
	The following command line shows the format for invoking a command procedure in the library: @LB:[1,2]INDSYS/LB:procedurename Before you attempt to access a command procedure, make sure that INDSYS.CLB is in LB:[1,2]. If it is not in this directory, your system manager must copy the library from the source kit for the system.
/[NO]LO	Indicates that when a new command file is executing, it can have access to the local symbols created by its calling command file and that any local symbols created by the new command file will be defined as local symbols for the calling command file. The default is /NOLO.
/[NO]DE	Indicates that the indirect command file is to be deleted when its processing is complete unless a logical end-of-file (/) or .STOP directive is encountered before the end of the file. The default is /NODE.

You may use any combination of the switches in the command line @filespec/switch(es) or in the directive statement .CHAIN filespec/switch(es). Except for /LB and /LO, the switches you specify in the command line that initiates Indirect processing are used as defaults when executing those commands.

2.6 Description of Indirect Directives

Directives must be separated from their arguments and from CLI-specific commands by at least one space. Unless you are using the .IF directives, only one directive is allowed on each command line.

You can insert any number of blanks and horizontal tabs in three places in a command line:

- At the start of the command line
- Immediately following the colon (:) of a label
- At the end of the command line

This allows you to format the command files so that they can be read easily. The recommended procedure is to begin labels in the first column and everything else in the ninth column (after one horizontal tab).

An important exception are the lines processed between .ENABLE and .DISABLE DATA directives; no blanks or tabs are removed from these lines. For example:

```
        .IFT Z .GOTO 10

    .
    .
    .
.10:    .OPEN DATFIL
        .DATA XXXXX
.ENABLE DATA
This is data
that goes into
the data file.
.DISABLE DATA
        .GOTO 20
```

Note that the .DISABLE DATA statement must begin in the first column or Indirect will place it in the data file. You can also use the .CLOSE directive in place of .DISABLE DATA. It too must begin in the first column.

2.6.1 Define a Label

Labels always appear at the beginning of the line. They may be on a line with additional directives and/or a CLI command, on a line with a comment, or on a line by themselves. When control passes to a line with a label, the line is processed from the first character after the colon.

Commands do not have to be separated from the label by a space. Only one label is permitted on each line. Labels are one to six characters in length and must be preceded by a period and terminated with a colon. A label may contain only alphanumeric characters and/or dollar signs ($).

It is also possible to define a label as a direct-access label; once the label is found, its position in the command file is saved. This allows subsequent jumps to frequently called labels or subroutines to be effected quickly. The first statement processed after a jump to a direct-access label is the one on the next line.

The maximum number of direct-access labels you can define within an indirect command file depends on the version of the Indirect task you are using. (The maximum number is specified in the task-build file.) If you define more than the maximum number of labels allowed, the subsequent direct-access labels replace the earliest, and so on. The smaller the number of direct-access labels, the larger the amount of free space in the symbol table.

If you have a large command file that branches from a line to a label before that line, using direct-access labels can result in a substantial saving of processing time. Normally, Indirect searches for the label in every line below the one where the branch occurred. If the label is not found, Indirect wraps around to the top of the file to continue the search. With direct-access labels, however, Indirect can go immediately to the label.

To declare a label for direct access, leave the line following the colon blank.

Example

```
.100:   .ASK A Do you want to continue
        .IFT A .GOSUB 200
          .
          .
          .
.200:
        .;THIS IS THE START OF A SUBROUTINE
          .
          .
          .
        .RETURN
```

In this example, .200: is a direct-access label while .100: is not.

.label:

The target label of a .GOTO branch from within a Begin-End block must be contained in that block because the .GOTO directive cannot branch into another block. The target label of a .ONERR directive must also be contained on the same Begin-End level. The target label of a .GOSUB call from within a Begin-End block, however, can be outside the current block because program control returns to the block from which the .GOSUB call was made. For more information, see the descriptions of the .BEGIN, .GOSUB, .GOTO, and .ONERR directives (Sections 2.6.5, 2.6.17, 2.6.18, and 2.6.22, respectively.)

2.6.2 Ask a Question and Wait for a Reply

The .ASK directive displays a question on the terminal, waits for a reply, and sets a specified logical symbol to the value of true or false, depending on the reply. If the symbol has not already been defined, Indirect makes an entry in the symbol table. If the symbol has been defined, Indirect resets its value (true or false) in accordance with the reply. Indirect exits with a fatal error if the symbol was previously defined as a string or numeric symbol.

Formats (brackets are required syntax)

.ASK ssssss txt-strng
.ASK [default:timeout] ssssss txt-strng
.ASK [:timeout] ssssss txt-strng

Parameters

ssssss
The 1- to 6-character symbol to be assigned a true or false value.

txt-strng
The question or prompt that Indirect displays.

default
The default response; used if the question is answered with an empty line (null) or if timeout occurs. The default can be <TRUE> or <FALSE> or another logical variable or expression.

timeout
The timeout count. Indirect waits this long for a response, then applies the default answer. The format for timeout is nnu, where nn is the decimal number of time units to wait and u is S (seconds), M (minutes), or H (hours). The timeout count is valid only if timeout mode is enabled (.ENABLE TIMEOUT; see Section 2.6.12).

The entire .ASK statement must fit on one command line.

Note that if you omit the default value but specify a timeout count, the colon is required for positional identification.

When executing a .ASK directive, Indirect displays (unless .DISABLE DISPLAY is in effect) txt-strng prefixed by an asterisk (*) and suffixed with "? [Y/N]:". Indirect recognizes five answers:

Y RET	Set symbol ssssss to true.
N RET	Set symbol ssssss to false.
RET	Set symbol to false or to user-specified default value. The RET symbol indicates the RETURN key.

.ASK

ESC	Set symbol ssssss to true and set the special logical symbol <ESCAPE> to true only if escape recognition has been enabled. The ESC symbol indicates the ESCAPE or ALTMODE key.
CTRL/Z	If Control-Z mode is enabled, set <EOF> to true and proceed, else exit immediately.

Example

The directive statement

```
.ASK PRINT Do you want to print the file
```

displays

```
* Do you want to print the file? [Y/N]:
```

on the terminal. Symbol PRINT will be set to true or false after you type Y or N, or press the RETURN key or the ESCAPE key (if escape recognition is enabled).

2.6.3 Ask for Definition of a Numeric Symbol

The .ASKN directive displays on the terminal a request for a numeric value, waits for it to be entered, optionally tests the range for the numeric response and/or applies a default value, and sets the specified symbol accordingly. If the symbol has not previously been defined, Indirect makes an entry in the symbol table. If the symbol has already been defined, Indirect resets its value in accordance with the reply. Indirect exits with a fatal error if the symbol was previously defined as a logical or string symbol.

Formats (brackets are required syntax)

.ASKN ssssss txt-strng
.ASKN [low:high:default:timeout] ssssss txt-strng

Parameters

ssssss
The 1- to 6-character symbol to be assigned a numeric value.

txt-strng
The question or prompt that Indirect displays.

low:high
A numeric expression or symbol giving the value range for the response.

default
A numeric expression or symbol giving the default value by allowing it to time out or by pressing the RETURN key.

timeout
The timeout count. Indirect waits this long for a response, then applies the default answer. The format for timeout is nnu, where nn is the decimal number of time units to wait and u is S (seconds), M (minutes), or H (hours). The timeout count is valid only if timeout mode is enabled (.ENABLE TIMEOUT; see Section 2.6.12).

The entire .ASKN statement must fit on one command line.

Note that if you omit any of the parameters within the square brackets, any preceding colons are required for positional identification.

The command line cannot exceed 132_{10} characters in length. When executing a .ASKN directive, Indirect displays (unless .DISABLE DISPLAY is in effect) txt-strng prefixed by an asterisk (*) and suffixed with [O]: to indicate that the response will be taken as octal or with [D]: to indicate that the response will be taken as decimal. The reply must be a number either within the specified range or in the range 0 to 177777_8 (by default) or 0 to $65{,}535_{10}$.

If the response is outside the specified range, the following message is displayed:

```
AT. -- Value not in range
```

Indirect then repeats the query.

.ASKN

If an arithmetic operation yields a result greater than 177777_8 when computing the actual value of any of the arguments low, high, or default, a fatal error occurs and the following message is displayed:

```
AT. -- Numeric under- or overflow
```

If the response is an empty line (null) and a default value (default) was not specified, Indirect applies a default of 0. Note that in this case, the range, if specified, must include 0.

The response may be either octal or decimal; a leading number sign (#) forces octal, a trailing period (.) forces decimal. In the absence of either, Indirect applies a default radix. The default radix is decimal if either the range or default values are decimal expressions (followed by a period). Otherwise, the default radix is octal (unless decimal mode has been enabled). Indirect displays the default type as either [O] or [D].

To force a default decimal radix without specifying a range argument, use the following construction:

```
.ASKN [::0.] A Enter value
```

or

```
.ENABLE DECIMAL
.ASKN A Enter value
```

Examples

The directive statement

```
.ASKN SYM Define numeric symbol A
```

displays the following on the terminal:

```
* Define numeric symbol A [0]:
```

In this example, [O] is the default radix (octal).

Indirect then defines symbol SYM according to the reply entered.

In the next example, the directive statement

```
.ASKN [2:35:16:20S] NUMSYM Define numeric symbol A
```

displays the following on the terminal:

```
* Define numeric symbol A [0 R:2-35 D:16 T:20S]:
```

The format used in this display is as follows:

```
[x R:low-high D:default T:timeout],
```

where:

x	O if the default radix is octal or D if it is decimal.
R:low-high	The specified range.
D:default	The specified default.
T:timeout	The specified timeout count before the default answer is used.

Indirect then checks whether the response string is in the specified range.

In the next example, the directive statement

```
.ASKN [NUMSYM+10:45:NUMSYM+10] SYM Define numeric symbol B
```

displays the following on the terminal (assuming the value of 16_8 for NUMSYM):

```
* Define numeric symbol B [0 R:26-45 D:26]:
```

.ASKS

2.6.4 Ask for Definition of a String Symbol

The .ASKS directive displays on the terminal a request for a string value to define a specified symbol and optionally tests whether the number of characters in the response string falls within the specified range. If the symbol has not previously been defined, Indirect makes an entry in the symbol table. If the symbol has already been defined, Indirect resets its value in accordance with the reply. Indirect exits with a fatal error if the symbol was defined previously as a logical or numeric symbol. If the number of characters is out of the specified range, the following message is displayed:

```
AT. -- String length not in range
```

Indirect then repeats the query.

Formats (brackets are required syntax)

.ASKS ssssss txt-strng
.ASKS [low:high:default:timeout] ssssss txt-strng

Parameters

ssssss
The 1- to 6-character symbol to be assigned a string value.

txt-strng
The prompt that Indirect displays.

low:high
A numeric expression giving the range for the number of characters permitted in the response string.

default
A string expression or symbol giving the default value.

timeout
The timeout count. Indirect waits this long for a response, then applies the default answer. The format for timeout is nnu, where nn is the decimal number of time units to wait and u is S (seconds), M (minutes), or H (hours). The timeout count is valid only if timeout mode is enabled (.ENABLE TIMEOUT; see Section 2.6.12).

The entire .ASKS statement must fit on one command line.

Note that if you omit any of the parameters within the square brackets, any preceding colons are required for positional identification.

When executing a .ASKS directive, Indirect displays (unless .DISABLE DISPLAY is in effect) txt-strng prefixed by an asterisk (*) and suffixes it with [S]:. The reply must be an ASCII character string.

Examples

The directive statement

```
.ASKS NAME Please enter your name
```

displays the following on the terminal:

```
* Please enter your name [S]:
```

Indirect then defines symbol NAME according to the string reply entered.

In the next example, the directive statement

```
.ASKS [1:15::10S] MIDNAM Please enter your middle name
```

displays the following on the terminal:

```
* Please enter your middle name [S R:1-15 T:10S]:
```

The format used in this display is as follows:

```
[S R:low-high T:timeout]
```

where:

S	The symbol type (string).
R:low-high	The specified range for the number of characters.
T:timeout	The specified timeout count.

.BEGIN

2.6.5 Begin Block

The .BEGIN directive marks the beginning of a Begin-End block. The block must be terminated with a .END directive.

Labels and local symbols defined following the .BEGIN directive are local to the block instead of being used throughout the entire command file. Therefore, labels and local symbols defined inside a block lose definition outside the block.

Symbols defined outside a block retain their definition within the block as well as throughout the file. Symbols defined outside a block and then modified within the block, however, assume and retain the value assigned in the block.

Labels defined outside a block are not accessible by a .GOTO directive from within the block. They are, however, accessible by a .GOSUB directive because program control returns to the next line within the block.

Labels and local symbols defined within a block lose definition with a .ERASE LOCAL directive statement (see Section 2.6.14) or with the .END directive.

The .BEGIN directive must be the only directive on a command line. For example, it cannot appear on the same line as a .IF directive.

Format

.BEGIN

2.6.6 Continue Processing Using Another File

The .CHAIN directive, which must be the last command in the file, closes the current file, erases all local symbols, clears any .ONERR arguments, empties the direct-access label cache, and continues processing using command lines from another file. The .CHAIN directive does not close data files, pass parameters, or change the nested file level.

Format (brackets not part of syntax)

.CHAIN filespec[/switch(es)]

Parameters

filespec

The specification (including a directory, if desired) of the file that contains the new command lines.

This parameter can also be a logical name assignment that translates into a valid FCS file specification.

/switch(es)

Any of the optional switches described in Section 2.5.

Example

```
.CHAIN OUTPUT
```

This directive statement transfers control to the file OUTPUT.CMD.

```
.CHAIN TEMP
```

This directive statement transfers control to the command file specified by the logical translation of TEMP.

```
.CHAIN OUTPUT
```

When the following DCL command lines are entered at the terminal

```
$ ASSIGN OUTPUT DB1:[TEST]OUTPUT.CMD [RET]
$ @TEST [RET]
```

Indirect transfers control to the command file specified by the translation of the logical name OUTPUT (DB1:[TEST]OUTPUT.CMD).

.CLOSE

2.6.7 Close Secondary File

The .CLOSE directive closes the secondary file opened by a .OPEN directive (see Section 2.6.23).

Format (brackets not part of syntax)

.CLOSE [#n]

Parameter

n

An optional file number in the range 0 to x-1, where x is the number of file-open FDBs specified in the build file for the Indirect task. (The value x is the maximum number of files that can be open simultaneously.) The default is #0. You can substitute a numeric symbol for the value n by enclosing the symbol in apostrophes.

2.6.8 Output Data to Secondary File

The .DATA directive specifies text that is to be output to a secondary file previously opened by a .OPEN directive.

When Indirect processes the text string that follows the .DATA directive, it ignores the leading space (if present), assuming it to be a separator between the directive and the text string. Any other spaces are transferred to the data file. If a tab follows the directive, it is transferred to the file. If no other characters follow the directive, a blank line is transferred to the file. This processing has the following results:

Command File	Open File
.DATA foo[RET]	foo[RET]
.DATA foo[RET]	foo[RET]
.DATA [TAB] foo[RET]	[TAB] foo[RET]
.DATA [TAB] foo[RET]	[TAB] foo[RET]
.DATA[RET]	null line

Note that if a comment follows a .DATA statement (that is, .DATA data !comment), Indirect also outputs the comment to the secondary file because it cannot tell if the comment pertains to the .DATA statement itself or to the data being output to the file.

Format (brackets not part of syntax)

.DATA [#n] txt-strng

Parameters

n

An optional file number in the range 0 to x-1, where x is the number of file-open FDBs specified in the build file for the Indirect task. (The value x is the maximum number of files that can be open simultaneously.) The default is #0. You can substitute a numeric symbol for the value n by enclosing the symbol in apostrophes.

txt-strng

The text to be output to the secondary file.

The command line cannot exceed 132_{10} characters and the specified text string cannot continue onto the next line. If a secondary file is not open, an error condition exists; Indirect issues an error message and begins error processing.

Example

```
.SETS SEND "This is data"
.OPEN TEMP
.DATA 'SEND'
.CLOSE
```

These directives output THIS IS DATA to the secondary file TEMP.DAT (DAT is the default file type for a data file).

.DEC

2.6.9 Decrement Numeric Symbol

The .DEC directive decrements a numeric symbol by 1. Indirect exits with a fatal error if the symbol was defined previously as a logical or string symbol.

Format

.DEC ssssss

Parameter

ssssss

The 1- to 6-character numeric symbol.

Example

```
.DEC X
```

This directive decrements by 1 the value assigned to the numeric symbol X. If X crosses the zero boundary (goes from positive to negative), decrementing it will cause an underflow error.

2.6.10 Delay Execution for a Specified Period of Time

The .DELAY directive delays further processing of the file for a specified period of time.

Format

.DELAY nnu

Parameters

nn

The decimal number of time units to delay.

u

T — Ticks

S — Seconds

M — Minutes

H — Hours

The parameter nn is decimal by default, or octal if preceded by a number sign (#). For example:

10S = 10_{10} seconds

#10S = 10_8 seconds

If quiet mode is disabled when the .DELAY directive is executed, Indirect issues the following message:

```
AT. -- Delaying
```

When the time period expires and the task resumes, Indirect issues this message:

```
AT. -- Continuing
```

The maximum amount of time you can specify for the .DELAY directive is 24 hours.

Example

```
.DELAY 20M
```

This directive statement delays processing for 20_{10} minutes.

.DISABLE

2.6.11 Disable Option

The .DISABLE directive disables a specified operating mode previously activated by a .ENABLE directive. See Section 2.6.12 for information on the operating modes.

Format

.DISABLE option[,option...]

Parameter

option

One or more of the operating modes described in Section 2.6.12.

The following is a list of the operating modes that can be disabled:

ATTACH	DELETE	GLOBAL	SUBSTITUTION
CLI	DETACH	LOWERCASE	TIMEOUT
CONTROL-Z	DISPLAY	MCR	TRACE
DATA	ESCAPE	OVERFLOW	TRUNCATE
DECIMAL	ESCAPE-SEQ	QUIET	

Note that when you disable DETACH mode from a command file and then request a task or CLI command to display information, the command file may not be able to continue executing. The task or CLI command may need to attach to the terminal to display the information, but will not be able to do so because Indirect cannot detach from the terminal.

2.6.12 Enable Option

The .ENABLE directive is used to invoke several operating modes. Each mode is independent of the others; all of them can be active simultaneously. When Indirect starts to process the highest-level command file, the initial settings are as follows:

ATTACH	enabled (R)	GLOBAL	disabled (I)
CLI	enabled (R)	LOWERCASE	enabled (I)
CONTROL-Z	disabled (I)	MCR	enabled (R)
DATA	disabled (I)	OVERFLOW	disabled (R)
DECIMAL	disabled (R)	QUIET	disabled (R)
DELETE	disabled (I)	SUBSTITUTION	disabled (I)
DETACH	enabled (R)	TIMEOUT	enabled (R)
DISPLAY	enabled (R)	TRACE	disabled (I)
ESCAPE	disabled (I)	TRUNCATE	disabled (R)
ESCAPE-SEQ	disabled (R)		

However, when Indirect passes control to a lower-level command file by means of a .CHAIN filename or @filename statement, only the following modes are reset to their initial (denoted by "I" in the previous list) settings: CONTROL-Z, DATA, DELETE, ESCAPE, GLOBAL, LOWERCASE, SUBSTITUTION, and TRACE. The remaining operating modes retain (denoted by "R" in the previous list) their new settings in the lower-level file.

In **ATTACH** mode, Indirect attaches to a terminal when displaying comment lines. In **DETACH** mode, it detaches from the terminal when processing command lines. Enabling both of these modes allows you to press CTRL/O to suppress a lengthy comment.

Attach and detach modes perform conditional IO.ATT and to IO.DET terminal-QIO functions, depending on the setting of the ATTACH or DETACH attribute bits in an internal flag word that controls the operating modes. However, disabling detach mode always attaches the terminal until a CLI command is issued or a .ENABLE QUIET statement is encountered (see the following description). Thus, if you want the terminal to remain detached while quiet mode is in effect, enable both attach and detach modes at the beginning of your indirect command file or interactive terminal session. Also, if you are going to toggle in the attach and detach operating modes during the execution of the command file, follow the .ENABLE/.DISABLE statement with a .ENABLE QUIET statement.

Enabling **CONTROL-Z** mode allows a command file to detect a CTRL/Z response to a question and continue processing. If Control-Z mode is disabled and you press CTRL/Z in response to a .ASKx question, Indirect exits. If Control-Z mode is enabled, the special symbol <EOF> is set to true and Indirect continues processing the command file.

In **DATA** mode, Indirect outputs lines that follow a .ENABLE DATA directive statement to a secondary file. (In contrast, the .DATA directive sends a single line of text to a secondary file.) To disable data mode, the .DISABLE DATA (or .CLOSE) statement must begin in the first

column. Otherwise, Indirect copies the statement itself into the data file. The .ENABLE DATA directive also has an optional argument (#n) that specifies which file the data is to go into. See the description of the .DATA directive (Section 2.6.8) for more information.

In **GLOBAL** symbol mode, symbol names that begin with a dollar sign ($) are defined as global to all levels of indirect files; once such a symbol has been defined, all levels recognize it. Symbols that do not begin with a dollar sign are recognized only within the level that defines them.

In **DECIMAL** mode, all numeric symbols are created or redefined by default as decimal instead of octal.

In **DELETE** mode, the current command file is deleted when Indirect processes the last command line in the file.

In **DISPLAY** mode, Indirect displays the current fields for the .ASKx directive and @ <EOF> . If display mode is disabled, Indirect displays only the text string for the .ASKx directive and suppresses @ <EOF> .

In **CLI** mode, commands not processed by Indirect are passed to your CLI. The CLI might be MCR, DCL, or a user-written CLI. CLI mode is equivalent to the function of the /CLI switch (see Section 2.5).

In **MCR** mode, commands not processed by Indirect are passed to your CLI. The CLI might be MCR, DCL, or a user-written CLI. MCR mode is equivalent to the function of the /MC switch (see Section 2.5).

In **LOWERCASE** mode, characters read from the terminal in response to .ASKS directives are stored in the string symbol without lowercase-to-uppercase conversion. The representation of characters is significant when comparing strings (see Section 2.6.19) because the .IF directive distinguishes between lowercase and uppercase characters.

In **SUBSTITUTION** mode, Indirect substitutes a string for a symbol. The symbol must begin and end in apostrophes ('symbol'). For example, if the symbol A has been assigned the string value THIS IS A TEST, then every 'A' will be replaced by THIS IS A TEST. When substitution mode is enabled, Indirect performs substitutions in each line before scanning the line for directives and CLI commands. (While obeying a .GOTO label directive, however, Indirect ignores any undefined symbols encountered before the target line, that is, the line containing the specified label.) You can also type the shorter **SUB** in place of SUBSTITUTION.

ESCAPE recognition permits the response to a .ASK, .ASKN, or .ASKS directive to be an escape character. A question answered with a single escape character sets the special logical symbol <ESCAPE> to true. The escape character must be used only as an immediate terminator to the question; if one or more characters precede the escape character, an error condition exists. In this case, the following message is displayed:

```
AT. -- Invalid answer or terminator
```

Indirect then repeats the question. Note that if you press the ESCAPE key in response to a .ASK directive, the specified logical symbol (ssssss of .ASK ssssss txt-strng) is also set to true.

.ENABLE

ESCAPE-SEQuence processing forces Indirect to attach to the terminal for escape-sequence recognition, using the IO.ATT!TF.SEQ I/O function. In this mode, the result of a .ASKx or .READ statement from the terminal will contain the terminating escape character and escape sequence as documented in the chapter on the full-duplex terminal driver in the *RSX-11M-PLUS and Micro/RSX I/O Drivers Reference Manual.*

OVERFLOW mode allows signed arithmetic in numeric expressions. When you enable overflow mode, Indirect evaluates numeric expressions as signed integers rather than as unsigned integers. Enabling this mode provides for numeric expressions and operations that otherwise would result in the "Numeric under- or overflow" error message.

In **QUIET** mode, Indirect does not echo CLI command lines or comments. The command lines are executed normally and, if they return a message or display, the message or display is shown on the terminal.

In **TIMEOUT** mode, Indirect uses the timeout parameters specified with the .ASKx directives. Indirect waits for the timeout count to elapse and then applies the default answer to the directives. Timeout mode must be enabled (the default) to use the timeout counts for the .ASKx directives.

In **TRACE** mode, command lines that Indirect has processed are displayed on the terminal. As each line is processed, it is displayed with its nesting level and an exclamation point (!). Trace mode is equivalent to the function of the /TR switch (see Section 2.5).

In **TRUNCATE** mode, Indirect ignores any truncation errors on a .READ directive. A truncation error occurs when a line in a file is too long. If the full record cannot fit within the 132_{10}-character limit of the symbol, the record is truncated.

Formats (brackets not part of syntax)

```
.ENABLE   option[,option...]
.ENABLE DATA  [#n]
```

Parameters

option

One or more of the operating modes described previously.

#n

An optional file number in the range 0 to x-1, where x is the number of file-open FDBs specified in the build file for the Indirect task. (The value x is the maximum number of files that can be open simultaneously.) The default is #0. You can substitute a numeric symbol for the value n by enclosing the symbol in apostrophes.

.ENABLE

Examples

SUBSTITUTION mode:

```
.ENABLE SUBSTITUTION
.ASKS FILE Specify next file
PRINT 'FILE'
```

While the command file is executing, the corresponding lines displayed at the terminal are:

```
$ * Specify next file [S]: SOURCES [RET]
$ PRINT SOURCES
```

GLOBAL symbol mode:

The following two lines appear in an indirect command file called TEST1:

```
.ENABLE GLOBAL
.SETS $X "TEST"
```

A file called TEST2.CMD contains the following lines:

```
.ENABLE GLOBAL
.ENABLE SUBSTITUTION
@TEST1
RUN '$X'
```

The CLI (in this case, MCR) displays the following lines when the file TEST2.CMD is run:

```
>RUN TEST
>@ <EOF>
```

ESCAPE-recognition mode:

```
        ;If you want a list of options, type <ESC>.
        .ENABLE ESCAPE
        .ASKS A Enter option
        .IFT <ESCAPE> .GOTO LIST
             .
             .
             .
.LIST:  ;Options are: A (ADD), S (SUBTRACT), M (MULTIPLY)
        .ASKS A Enter option
```

If you press the ESCAPE key in response to ENTER OPTION, the lines displayed at the terminal are:

```
>;If you want a list of options, type <ESC>.
>* Enter option [S]: [ESC]
>;Options are: A (ADD), S (SUBTRACT), M (MULTIPLY)
>* Enter option [S]:
```

QUIET mode (DCL is the CLI for the terminal):

```
.ASK QUIET Do you want command lines suppressed
.IFT QUIET .ENABLE QUIET
.IFF QUIET .DISABLE QUIET
SHOW TASKS/ACTIVE
```

If the response is affirmative, Indirect displays the active tasks but not the SHOW TASKS/ACTIVE command. For example:

```
$ * Do you want command lines suppressed? [Y/N]: Y
DCL
SHOT14
AT.T14
```

CONTROL-Z mode:

```
.ENABLE CONTROL-Z
.ASK RESP Do you want to continue
.IFT <EOF> .GOTO CLENUP
.IFF RESP .GOTO CLENUP
```

If you press CTRL/Z in response to the question, <EOF> is set to true and Indirect transfers to label CLENUP.

.END

2.6.13 End Block

The .END directive marks the end of a Begin-End block. If Indirect encounters more .END directives than .BEGIN directives, command processing terminates and the following message is displayed:

```
AT. -- Illegal nesting
```

Format

.END

As with .BEGIN, the .END directive must be the only directive on the command line.

2.6.14 Delete Symbols

The .ERASE directive deletes all local or global symbol definitions, or a specific local or global symbol definition.

When you define a symbol either locally (by defining a symbol value) or globally (by enabling global symbol mode and preceding the symbol name with a dollar sign ($)), Indirect creates an entry in the symbol table. The .ERASE directive erases either all local or all global entries, or a specific local or global entry, in the table.

Following a .ERASE directive, you can redefine a symbol's value as well as its type.

Formats

```
.ERASE LOCAL
.ERASE GLOBAL
.ERASE SYMBOL   symbol
```

A .ERASE LOCAL directive outside of a Begin-End block erases all local symbols defined within the current file.

A .ERASE LOCAL directive within a Begin-End block erases only those local symbols defined within the block.

However, note that the following actions also occur:

1. Local symbols defined within a nested file are erased when that file exits.
2. Local symbols defined within a Begin-End block are erased by .END.
3. Local symbols defined outside of Begin-End blocks are visible, modifiable, and not erasable within a Begin-End block.

A .ERASE GLOBAL directive, either outside of or within a Begin-End block, erases all global symbols.

A .ERASE SYMBOL symbol directive erases the specified local or global symbol.

Examples

```
.ERASE LOCAL
```

This directive erases all local symbol definitions used in the indirect command file.

```
.ERASE SYMBOL $SWITC
```

This directive erases the single global symbol "$SWITC."

.EXIT

2.6.15 Exit Current Command File

The .EXIT directive terminates processing of the current command file or Begin-End block and returns control to the previous-level command file or, if the directive is executed within a block, to the line following the .END directive. If the directive is encountered at the uppermost indirect nesting level, Indirect exits and passes control to the CLI (see the .STOP directive, Section 2.6.34).

The .EXIT directive also allows you to optionally specify a value to copy into the special symbol <EXSTAT> .

Format (brackets not part of syntax)

.EXIT [value]

Parameter

value

An optional numeric expression to be copied to the special symbol <EXSTAT> .

Example

The following line appears in an indirect command file called TEST1:

```
@TEST2
```

The file TEST2.CMD contains the following line:

```
.EXIT
```

When Indirect encounters the .EXIT directive in TEST2, control returns to TEST1.CMD.

If the .EXIT directive in TEST2.CMD includes a numeric expression (for example, .EXIT N+2), Indirect evaluates the expression and copies the value into <EXSTAT> .

2.6.16 Interface to FMS-11

The .FORM directive provides access to the FMS-11 form driver. It allows FMS commands (see the following list) to be passed to FMS-11.

You must have a license and have installed the FMS-11 kit. You should be familiar with the documentation before attempting to use .FORM.

The FMS-11 support in Indirect includes support for the VT100 and VT200 terminals. However, you cannot use VT200 terminals unless they are in VT100 mode.

The syntax of the .FORM directive parallels the format of the MACRO-11 call interface to FMS-11. You are encouraged to read the *FMS-11/RSX Programmer's Guide,* particularly the chapters on form-driver operation and the MACRO-11 interface.

Format

.FORM command,p1,p2...,pn

Parameters

command

One of the .FORM commands. The commands are a subset of the commands (codes) used in the MACRO-11/FMS interface.

p1 to pn

Parameters for the command.

The following list gives the commands for the .FORM directive and their formats. Square brackets indicate optional parameters.

Command	Description	Format
ALL	Return all fields	.FORM ALL[,retval[,rettrm]]
ANY	Return any field value	.FORM ANY[,retnam[,retinx[,retvaL[,rettrm]]]]
CLS	Close forms library	.FORM CLS
CSH	Clear screen and show form	.FORM CSH,fldname[,linenum]
DAT	Get data from form	.FORM DAT,[fldname],[index],[retval]
GET	Get value for specified field	.FORM GET,fldname[,index[,retnam[,retinx [,retval[,rettrm]]]]]
GSC	Get current line of scrolled area	.FORM GSC,fldname,retval[,rettrm]
LST	Output to last line of screen	.FORM LST [,value]
OPN	Open forms library	.FORM OPN,filename
PAL	Put all fields	.FORM PAL [,value]
PSC	Put to current line of scrolled area	.FORM PSC,fldname,value

.FORM

Command	Description	Format
PUT	Put to specified field	.FORM PUT,fldname,[index,]value
RAL	Return all fields	.FORM RAL,value
RTN	Return value for specified field	.FORM RTN,fldname,[index,]retval
SHO	Show form	.FORM SHO,fldname[,linenum]
SPF/SPN	Supervisor mode control	.FORM SPF or .FORM SPN
TRM	Process field terminator	.FORM TRM,[fldname,][value,]terminator [,retnam[,retinx]]

The parameters for the .FORM commands are listed below.

Command Parameters

filename
The name of a string variable or constant naming the file in which the form definitions are stored.

fldname
The name of a field defined in the currently displayed form.

index
In an indexed field, the index referencing the specific field being addressed.

linenum
The screen line number where the form display is to begin.

retinx
The index of the field you complete.

retnam
The name of the field you complete.

retTrm
The name of a numeric variable to contain the code for the terminator that you specify.

retval
The name of a string variable into which the returned value will be placed.

terminator
The code for the terminator to be processed.

value
A string variable or constant to be placed in the indicated field.

String values supplied *to* the .FORM directive can be expressed as a constant enclosed in quotation marks or as the name of a previously defined string variable. For example:

```
.FORM OPN,"FMSDEM.FLB"
```

and

```
.SETS LIBR "FMSDEM.FLB"
.FORM OPN,LIBR
```

are equivalent.

String and numeric values returned *from* the .FORM directive are passed as though a .SETS or .SETN directive were being executed. This means that the name of the variable to receive the value must be supplied and that it must either have not been defined previously or is now defined as the appropriate string or numeric type. For example:

```
.FORM GET,"CHOICE",,,fldval
```

In this directive statement, the parameter fldval is defined or redefined as required and contains the string you type to fill the field named CHOICE on the currently displayed form.

A demonstration procedure is included in LB:[1,2]INDSYS.CLB. To execute the procedure, type the following command line:

```
@LB:[1,2]INDSYS.CLB/LB:FMSDEM [RET]
```

After the terminal type is verified, a copy of the forms library is placed in your directory. The procedure is identical to that provided in MACRO-11 form on the FMS-11 kit. Refer to the FMS documentation for more information on the demonstration.

.GOSUB

2.6.17 Call a Subroutine

The .GOSUB directive saves the current position in an indirect command file and then branches to a label. The label identifies an entry point to a subroutine that is terminated by a .RETURN directive.

When you issue a .GOSUB directive from within a Begin-End block, Indirect saves the current block context and then scans the file for the first occurrence of the subroutine label. Note that during the scan, Indirect ignores any intervening .BEGIN or .END directives. The .RETURN directive restores previous block context, so the subroutine can be contained within a Begin-End block.

The maximum nesting depth for subroutine calls depends on the number specified in the build file for the Indirect task.

Format

.GOSUB label parameters

Parameter

label

The label that designates the first line of a subroutine, but without the leading period and trailing colon. Any parameters to the right of the label and to the left of a comment are transferred to the reserved local symbol COMMAN. The value of COMMAN can then be parsed with the .PARSE directive (see Section 2.6.26) to obtain formal call parameters.

Example

```
.GOSUB EVAL
```

This directive statement transfers control to the subroutine labeled .EVAL:.

2.6.18 Branch to a Label

The .GOTO directive causes a branch from one line in an indirect command file to another line. All commands between the .GOTO directive and the specified label are ignored. Branches can go forward or backward in the file.

The target of a .GOTO branch from within a Begin-End block must be contained in that block. The .GOTO directive cannot branch into another block. When Indirect encounters a .GOTO directive within a Begin-End block, it searches for the specified label in that block. Since Indirect only searches the one Begin-End block, you can use the same label more than once in a command file.

See Section 2.6.1 for more information on labels and direct-access labels.

Format

.GOTO label

Parameter

label

The name of the label, but without the leading period and trailing colon.

Example

```
.GOTO 100
```

This directive statement transfers control to the line containing the label .100:.

.IF

2.6.19 Logical Test

A number of directives make tests. If the result of the test is true, Indirect processes the remainder of the command line. Logical tests can be combined into a compound logical test by using the .AND and .OR directives.

2.6.19.1 Test if Symbol Meets Specified Condition (.IF)

The .IF directive compares a numeric or string symbol with another expression of the same type to determine if one of several possible conditions is true. If the condition is satisfied, Indirect executes the remainder of the command line.

When comparing a string symbol with a string expression, Indirect compares the ASCII values of each operand's characters (from left to right) one by one. An operand is considered greater if the first nonequal character has a greater value than the corresponding character in the other operand.

Numeric symbols are compared strictly on the basis of magnitude. If overflow mode is enabled (see Section 2.6.12), Indirect evaluates numeric expressions as signed integers rather than as unsigned integers.

Format

.IF symbol relop expr directive-statement

Parameters

symbol
The 1- to 6-character logical, numeric, or string symbol.

relop
One of the following relational operators:

EQ or =	Equal to
NE or <>	Not equal to
GE or >=	Greater than or equal to
LE or <=	Less than or equal to
GT or >	Greater than
LT or <	Less than

expr
An expression of the same type as symbol.

directive-statement
The Indirect command line to be processed if the condition is satisfied.

Examples

```
.SETS X "A"
.SETS Y "a"
.IF X LT Y  .GOTO 200
```

The ASCII value of string symbol X is less than the ASCII value of string symbol Y, which satisfies the less-than condition. Thus, control passes to the line containing the label .200:.

```
.SETN N1 2
.SETN N2 7
.IF N1 <= N2  DIR
```

With the condition satisfied (numeric symbol N1 less than or equal to numeric symbol N2), the (DCL) DIRECTORY command is executed.

```
.SETS S1 "AAb"
.SETS S2 "AA"
.SETS S3 "BBBB"
.IF S1 >= S2+S3[1:1]  .INC A
```

Because string symbol S1 is greater than or equal to string symbol S2 concatenated with the first character of string symbol S3 (AAb > = AAB), that condition is satisfied and Indirect increments numeric symbol A.

2.6.19.2 Test If Task Is Active or Dormant (.IFACT/.IFNACT)

The .IFACT and .IFNACT directives test whether a task is active (.IFACT) or dormant (.IFNACT). If the result of the test is true, the remainder of the command line is processed. If the specified task is not installed, Indirect assumes the dormant condition.

Formats

.IFACT taskname directive-statement
.IFNACT taskname directive-statement

Parameters

taskname

A 1- to 6-character valid task name.

directive-statement

The Indirect command line to be processed if the condition is satisfied.

Examples

```
.IFACT   REPORT  .GOTO 350
.IFNACT  REPORT  RUN REPORT
```

2.6.19.3 Test if Symbol Is Defined or Not Defined (.IFDF/IFNDF)

The .IFDF and .IFNDF directives test whether a logical, numeric, or string symbol has been defined (.IFDF) or not defined (.IFNDF). If the result of the test is true, the remainder of the command line is processed. These directives do not test the value of the symbol.

The directives .IFT symb, .IFF symb, and .IF symb should not be used on the same line as the .IFDF symb directive. Because the .IFDF symb directive evaluates to false, Indirect processes the remainder of the command line looking for a .OR directive. Instead, it encounters .IFT symb, .IFF symb, or .IF symb, but because the symbol is undefined, an error message is generated.

The following example shows how to test whether a symbol is defined and how to then use that symbol:

```
.IFNDF symbol .GOTO 10$
.IFT symbol <action...>
   .
   .
   .
.10$:
.IFNDF symbol .SETF symbol
   .
   .
   .
```

Formats

.IFDF ssssss directive-statement
.IFNDF ssssss directive-statement

Parameters

ssssss
The 1- to 6-character symbol being tested. The symbol can be local, global, or an Indirect special symbol.

directive-statement
The Indirect command line to be processed if the condition is satisfied.

Examples

```
.IFDF  A  .GOTO 100
.IFNDF  A  .ASK A Do you want to set the time
```

2.6.19.4 Test if Task is Installed or Not Installed (.IFINS/.IFNINS)

The .IFINS and .IFNINS directives test whether a task is installed (.IFINS) or not installed (.IFNINS) in the system. If the result of the test is true, the remainder of the command line is processed.

Formats

.IFINS taskname directive-statement
.IFNINS taskname directive-statement

Parameters

taskname
A 1- to 6-character task name.

directive-statement
The Indirect command line to be processed if the condition is satisfied.

Examples

```
.IFINS  LP1  .GOTO 250
.IFNINS  LP1  INS $LP1
```

2.6.19.5 Test if Mode is Enabled or Disabled (.IFENABLED/.IFDISABLED)

The .IFENABLED and .IFDISABLED directives test whether an operating mode has been enabled with the .ENABLE directive or disabled with the .DISABLE directive. See the description of the .ENABLE directive in Section 2.6.12 for the list of operating modes.

Formats

.IFENABLED option directive-statement
.IFDISABLED option directive-statement

Parameters

option
The same operating mode option (with the exception of DATA) used with the .ENABLE or .DISABLE directive, or one of the following options:

FMS	The FMS-11/RSX form driver is present in the system; default is enabled.
FULL-DUPLEX	The full-duplex terminal driver is present in the system; default is enabled.
LOCAL	The /LO switch was specified in the initial command line; default is enabled.
POTASK	Parent/offspring tasking support is included in the current system; default is enabled.

directive-statement
The Indirect command line to be processed if the condition is satisfied.

Examples

```
.IFENABLED CLI .GOTO SHOW
.IFDISABLED DECIMAL .ENABLE DECIMAL
```

2.6.19.6 Test if Driver Is Loaded or Not Loaded (.IFLOA/.IFNLOA)

The .IFLOA and .IFNLOA directives test whether a driver is loaded (.IFLOA) or not loaded (.IFNLOA) in the system. If the result of the test is true, the remainder of the command line is processed. Note that for the purposes of these directives, resident drivers are assumed to be loaded.

Formats

.IFLOA dd: directive-statement
.IFNLOA dd: directive-statement

Parameters

dd:
A device driver

directive-statement
The Indirect command line to be processed if the condition is satisfied.

Examples

```
.IFLOA DU: .GOTO 250
.IFNLOA DU: LOA DU:
```

2.6.19.7 Test if Symbol Is True or False (.IFT/.IFF)

The .IFT and .IFF directives test whether a logical symbol is true (.IFT) or false (.IFF). If the result of the test is true, Indirect processes the remainder of the command line.

Indirect exits with a fatal error if the symbol being tested was previously defined as a numeric or string symbol.

Formats

.IFT ssssss directive-statement
.IFF ssssss directive-statement

Parameters

ssssss

The 1- to 6-character logical symbol being tested.

directive-statement

The Indirect command line to be processed if the condition is satisfied.

Examples

```
.IFT  A  .GOTO  100
.IFF  B  .GOTO  200
```

2.6.19.8 Compound Tests

You can combine .IF tests by using the .AND and .OR directives. In addition, an implied .AND is effected when more than one .IF appears on the same line without being separated by a .AND directive.

The .AND directive takes precedence over the .OR directive as shown in the following example:

```
.IFT A .OR .IFT B .AND .IFT C .GOTO D
```

That is, Indirect reads the line as:

```
.IFT A .OR (.IFT B .AND .IFT C) .GOTO D
```

Examples

```
.IFT A .AND .IFF B  .GOTO HELP
```

If the logical symbol A is true and the logical symbol B is false, control passes to the line containing the label .HELP:.

```
.IFT A .IFF B  .GOTO HELP
```

This has the same effect as the previous directive (.AND implied).

```
.IFT A .OR  .IFF B RUN WAY
```

If the logical symbol A is true or if the logical symbol B is false, the RUN command is issued.

.INC

2.6.20 Increment Numeric Symbol

The .INC directive increments a numeric symbol by 1. Indirect exits with a fatal error if the symbol was previously defined as a logical or string symbol.

Format

.INC ssssss

Parameter

ssssss

The 1- to 6-character numeric symbol being incremented.

Example

```
.INC B
```

This directive increments by 1 the value assigned to the numeric symbol B. If B crosses the zero boundary (goes from negative to positive), incrementing it will cause an overflow error.

2.6.21 Define Logical End-of-File

The logical end-of-file directive (/) terminates file processing at all levels, closes all open data files, and exits. Indirect then displays (if display mode has not been disabled) the following message:

```
@ <EOF>
```

Format

```
/
```

The directive is the first nonblank character of the line.

You can use this directive at any location in the command file to quickly terminate file processing, but care should be taken to avoid an inadvertent exit.

Example

```
        .ASK CONT Do you wish to continue
         .IFT CONT .GOTO 100
         /
.100:
```

.ONERR

2.6.22 Branch to Label on Detecting an Error

If Indirect detects one of the following errors, control passes to the line containing the label specified with the .ONERR directive:

- Task not installed in system (.XQT, .WAIT)
- Undefined symbol
- Bad syntax (.XQT, .WAIT, .DELAY)
- Unrecognized command
- String substitution error
- Symbol type error (.IF, .IFT, .IFF, .INC, .DEC)
- Redefinition of a symbol to a different type (.ASK, .ASKN, .ASKS, .SETT, .SETF, .SETL, .SETN, .SETD, .SETO, .SETS)
- Data file error (.OPEN, .OPENA, .OPENR, .DATA, .CLOSE, or .READ between .ENABLE DATA and .DISABLE DATA)

This feature provides you with a means of gaining control to terminate command file processing in an orderly manner.

Note that the .ONERR directive applies only to the error conditions listed; errors returned from a task external to Indirect (for example, a DCL syntax error) are not processed by the .ONERR directive.

Format (brackets not part of syntax):

.ONERR [label]

Parameter

label

The name of the label, but without the leading period and trailing colon.

Upon detecting an error, Indirect passes control to the line starting with .label:. The .ONERR directive must be issued before Indirect encounters the error condition. If the directive is executed (one of the listed errors is encountered), error processing passes to the specified label. If the label specified by the .ONERR directive does not exist and an error condition has occurred, command processing terminates.

If you do not specify the optional label, Indirect disables processing for the previous .ONERR directive.

If you want to have .ONERR processing and Begin-End blocks in a program, the label you specify must be located on the same block level as the .ONERR directive. When Indirect detects an error, it passes control to the most recently defined .ONERR label in the current block level or in a previous, lower block level.

Once a .ONERR condition has occurred, another .ONERR directive must be issued to trap a future error.

.ONERR

The .ONERR directive works with the special symbol <ERRCTL> (see Section 2.4.1.2). For each class of error that a .ONERR target routine processes, the appropriate bit is set in the symbol. The initial default value for <ERRCTL> is 1, which implies that only class 1 errors can be handled with a .ONERR routine. (Note that if you attempt to process errors other than default class 1, Indirect cannot continue in most cases. The error service routine is limited to a fatal error message and .EXIT. The internal state of Indirect is indeterminate in all but class 1 error cases.) After Indirect has processed the .ONERR directive and passed control to a .ONERR label, <ERRCTL> is reset to 1.

See Appendix A for a list of error messages and their assigned class values.

Example

```
.ONERR  100
```

Upon detecting one of the error conditions, Indirect passes control to the line labeled .100:.

.OPEN

2.6.23 Open Secondary File

The .OPEN directive opens a specified secondary file as an output file. The special symbol <FILERR> is assigned the resulting FCS-11 or directive status code. The .DATA directive is used to place data in the secondary file opened by the .OPEN directive.

Format (brackets not part of syntax)

.OPEN [#n] filespec

Parameters

#n

An optional file number in the range 0 to x-1, where x is the number of file-open FDBs specified in the build file for the Indirect task. (The value x is the maximum number of files that can be open simultaneously.) The default is #0. You can substitute a numeric symbol for the value n by enclosing the symbol in apostrophes.

filespec

A file to be opened as an output file. The default file type is DAT.

Indirect sets the owner UIC of the file being opened to be the current protection UIC of the user. All FCS protection and privilege checks are still in effect.

For nonprivileged users, the protection UIC is always the same as their login UIC. If you are not in named directory mode, you can change only your default UFD with the MCR SET /UIC or SET /DEF or DCL SET UIC or SET DEF commands. If you are in named directory mode, the SET /UIC and SET UIC commands are invalid, and SET /DEF and SET DEFAULT change only your default UFD.

For privileged users, the protection UIC can change. If you are not in named directory mode, you can use SET /UIC and SET /DEF (or their equivalent DCL commands) to change both your protection UIC and your default UFD. If you are in named directory mode, the SET /UIC (or SET UIC) command changes only your protection UIC. Your default UFD remains the same. The SET /DEF (or SET DEF) command changes only your default UFD. Your protection UIC remains the same.

For more information on named directories and UICs, see the *RSX-11M-PLUS MCR Operations Manual* or the *RSX-11M-PLUS Command Language Manual.*

The parameter filespec can also be a logical name assignment that translates into a valid FCS file specification.

You cannot specify a fixed-length record file with the .OPEN directive. If you do, Indirect changes the attribute of the file from fixed-length to variable-length when it closes the file.

Note that you cannot include a comment that begins with a semicolon (;comment) in a .OPEN statement. Doing so results in a syntax error. Comments that begin with an exclamation point (!comment) are accepted.

Examples

```
.OPEN SECOUT
```

This directive opens the file SECOUT.DAT as an output file.

```
.OPEN TEMP
```

This directive opens the file specified by the logical translation of TEMP.

The command file HIHO.CMD contains the following directive statement:

```
.OPEN GRUMPY
```

When the following command lines are executed:

```
$ ASSIGN DU2:[DWARFS]GRUMPY.DAT GRUMPY [RET]
$ @HIHO [RET]
```

Indirect opens the file DU2:[DWARFS]GRUMPY.DAT as an output file.

.OPENA

2.6.24 Open Secondary File for Append

The .OPENA directive opens a secondary file and appends all subsequent data to the file.

Format (brackets not part of syntax)

.OPENA [#n] filespec

Parameters

n

An optional file number in the range 0 to x-1, where x is the number of file-open FDBs specified in the build file for the Indirect task. (The value x is the maximum number of files that can be open for append simultaneously.) The default is #0. You can substitute a numeric symbol for the value n by enclosing the symbol in apostrophes.

filespec

A secondary file to be opened with subsequent data appended to it. The default file type is DAT.

Indirect sets the owner UIC of the file being opened to the current protection UIC of the user. See the description of the .OPEN directive for more information.

The parameter filespec can also be a logical name assignment that translates into a valid FCS file specification.

You cannot specify a fixed-length record file with the .OPENA directive. If you do, Indirect changes the attribute of the file from fixed-length to variable-length record when it closes the file.

Note that you cannot include a comment that begins with a semicolon (;comment) in a .OPENA statement. Doing so results in a syntax error. Comments that begin with an exclamation point (!comment) are accepted.

If the specified file does not already exist, .OPENA becomes the .OPEN directive by default.

Examples

```
.OPENA SECOUT
```

This directive opens the file SECOUT.DAT as an output file and appends subsequent data to it.

```
.OPENA TEMP
```

This directive opens the file specified by the logical translation of TEMP as an output file and appends subsequent data to it.

The command file BEAUTY.CMD contains the following directive statement:

```
.OPENA BEAST
```

When the following DCL command lines are executed:

```
$ ASSIGN BEAST DU2:[TALES]BEAST.DAT [RET]
$ @BEAUTY [RET]
```

Indirect opens the file DU2:[TALES]BEAST.DAT and appends subsequent data to it.

.OPENR

2.6.25 Open File for Reading

The .OPENR directive opens a file for reading with the .READ directive.

Format (brackets not part of syntax)

.OPENR [#n] filespec

Parameters

n

An optional file number in the range 0 to x-1, where x is the number of file-open FDBs specified in the build file for the Indirect task. (The value x is the maximum number of files that can be open for reading simultaneously.) The default is #0. You can substitute a numeric symbol for the value n by enclosing the symbol in apostrophes.

filespec

A file to be opened for reading. The default file type is DAT.

Indirect sets the owner UIC of the file being opened to the current protection UIC of the user. See the description of the .OPEN directive for more information.

The parameter filespec can also be a logical name assignment that translates into a valid FCS file specification.

You cannot specify a fixed-length record file with the .OPENR directive. If you do, Indirect changes the attribute of the file from fixed-length to variable-length record when it closes the file.

Note that you cannot include a comment that begins with a semicolon (;comment) in a .OPENR statement. Doing so results in a syntax error. Comments that begin with an exclamation point (!comment) are accepted.

Examples

```
.OPENR INDADD
```

This directive opens the file INDADD.DAT for reading with the .READ directive.

```
.OPENR DATLIB.ULB/LB:DATINP
```

This directive opens for reading the library module DATINP that is contained in the universal library DATLIB.

```
.OPENR TEMP
```

This directive opens for reading the file specified by the logical translation of TEMP.

The command file HANSEL.CMD contains the following directive statement:

```
.OPENR GRETEL
```

When the following DCL command lines are executed:

```
$ ASSIGN GRETEL DU2:[WITCH]GRETEL.DAT [RET]
$ @HANSEL [RET]
```

Indirect opens the file DU2:[WITCH]GRETEL.DAT for reading.

.PARSE

2.6.26 Parse Strings into Substrings

The .PARSE directive parses strings in a command line into substrings.

Format

.PARSE <string> <controlstring> <var1> <var2> ... <varn>

The string is broken up into substrings as specified by the control string. The substrings are stored in the specified variables. The first character of the control string delimits the first substring, the second character of the control string delimits the second substring, and so on. The last character of the control string is repeated if the number of variables exceeds the length of the control string. If you specify more variables than substrings, the additional variables are set to null strings. If you specify fewer variables than the number of substrings that can be parsed, the last variable contains the unparsed fragment of <string> .

If you specify only one variable, Indirect discards all characters following, and including, the delimiter (for example, a comma or a right angle bracket). All null substrings are also discarded. If you specify more than one variable and the last character of <string> is a delimiter, Indirect assumes that there is a null substring after it. If you do not specify a symbol for this substring to be parsed into, the delimiter and the substring are parsed into the last symbol specified.

The symbol <STRLEN> contains the actual number of substrings that Indirect processed (including explicit null substrings).

Examples

A command file, PARSE.CMD, contains the following command lines:

```
.ENABLE SUBSTITUTION
.PARSE COMMAN " ," FILE A1 A2 A3 A4 A5
;FILE = 'FILE'
;A1 = 'A1'
;A2 = 'A2'
;A3 = 'A3'
;A4 = 'A4'
;A5 = 'A5'
;<STRLEN> = '<STRLEN>'
```

The command file is invoked with the following command line:

```
>@PARSE ARG1,ARG2,,ARG3 [RET]
```

When the file is executed, COMMAN contains "PARSE ARG1, ARG2,,ARG3" and Indirect displays the following information:

```
>;FILE = PARSE
>;A1 = ARG1
>;A2 = ARG2
>;A3 =
>;A4 = ARG3
>;A5 =
>;<STRLEN> = 5
```

The following example is from an interactive terminal session:

```
>@ti: [RET]
AT.>.enable substitution [RET]
AT.>.sets a "1,2," [RET]
AT.>.parse a "," b c d [RET]
AT.>;'b' [RET]
>;1
AT.>;'c' [RET]
>;2                              (null substring)
AT.>;'d' [RET]
;
AT.>.parse a "," b c [RET]
AT.>;'b' [RET]
>;1
AT.>;'c' [RET]
>;2,
AT.>[CTRL/Z]
>@ <EOF>
>
```

In this example, the first time string A is parsed, there are enough variables to contain the substrings 1 and 2 and the implied null substring following the 2. The second time A is parsed, however, there are not enough variables to contain the substrings, so the first substring (1) is parsed into the first variable, and the second substring and the delimiter (2,) are parsed into the second variable.

.PAUSE

2.6.27 Pause for Operator Action

The .PAUSE directive interrupts processing of an indirect command file to wait for user action. A .PAUSE directive causes Indirect to stop itself, after which you can perform some operations and subsequently cause the task to resume.

Note that your terminal must be set to NOSERIAL before you execute any indirect command file that uses a .PAUSE directive. If you do not issue the MCR command SET/NOSERIAL or the DCL command SET TERMINAL/NOSERIAL command before executing an indirect command file that includes a .PAUSE directive, you will not be able to complete the START or UNSTOP command.

Format

.PAUSE

When Indirect stops itself, it displays the following message on the entering terminal:

```
AT. -- Pausing, To continue type "command taskname"
```

command
The command line to be issued to resume the task.

taskname
The name of the Indirect task.

You then type the appropriate command line to resume the task. Indirect displays the following message and continues processing where it left off:

```
AT. -- Continuing
```

Note that the .PAUSE directive is valid only if your command line interpreter is MCR or DCL.

2.6.28 Read Next Record

The .READ directive reads the next record into a specified string variable. The entire record is read into the variable. If the record is longer than 132_{10} characters, an error occurs.

After every .READ operation, the special symbol <FILERR> contains the FCS-11 file code for the read and the special symbol <EOF> reflects whether an end-of-file was found. (Note that .OPENR does not clear <EOF> .) If an error or end-of-file occurs, the string variable remains unchanged from its previous state.

Format (brackets not part of syntax)

.READ [#n] ssssss

Parameters

n

An optional file number that specifies the file from which the record is to be read. The file number must be one of the numbers used in a previous .OPENR statement.

ssssss

The string variable into which the record will be read.

Example

```
          .ENABLE SUBSTITUTION
          .OPENR FILE
          .IF <FILERR> NE 1 .GOTO ERROR
.LOOP:
          .READ RECORD
          .IFT <EOF> .GOTO DONE
          .IF <FILERR> NE 1 .GOTO ERROR
          ; 'RECORD'
          .GOTO LOOP
            .
.ERROR:     .
            .
.DONE:    .CLOSE
            .
            .
            .
```

These directives open the file FILE.DAT for reading, read each record into the string variable RECORD, display each record on the terminal, and close the file.

.RETURN

2.6.29 Return from a Subroutine

The .RETURN directive signifies the end of a subroutine and returns control to the line immediately following the .GOSUB directive that initiated the subroutine.

Format

.RETURN

2.6.30 Set Symbol to True or False

The .SETT, .SETF, and .SETL directives define or change the value of a specified logical symbol. If the symbol has not been defined, Indirect makes an entry in the symbol table and sets the logical symbol to the value specified. If the symbol has already been defined, Indirect resets the symbol accordingly. Indirect exits with a fatal error if the logical symbol was defined previously as a numeric or string symbol.

Formats

.SETT ssssss
.SETF ssssss
.SETL ssssss llllll

Parameters

ssssss

The 1- to 6-character logical symbol to be assigned a true or false value.

llllll

A logical or numeric expression. The symbol ssssss is assigned the value of llllll when the logical expression is evaluated.

Examples

```
.SETT X
```

This directive sets the logical symbol X to true.

```
.SETF ABCDE
```

This directive sets the logical symbol ABCDE to false.

```
.SETL TEST SWITCHA!SWITCHB
```

This directive sets the logical symbol TEST to true if SWITCHA or SWITCHB is true.

.SETN

2.6.31 Set Symbol to Numeric Value

The .SETN directive defines or changes the value of a specified numeric symbol. If the symbol has not been defined, Indirect makes an entry in the symbol table and sets the symbol to the numeric value specified. If the symbol has already been defined, Indirect resets the symbol accordingly. Indirect exits with a fatal error if the numeric symbol was previously defined as a logical or string symbol.

Format

.SETN ssssss numexp

Parameters

ssssss
The 1- to 6-character numeric symbol.

numexp
A numeric expression. (See Section 2.4.2.)

When specifying a numeric value to assign to a symbol, you may combine a numeric symbol or constant with another numeric symbol or constant to form a numeric expression. If numeric expressions are used, no embedded blanks or tabs are permitted. Evaluation is done from left to right unless parentheses are used to form subexpressions, which are evaluated first. The radix of an expression is octal if all the operands are octal and decimal mode has not been enabled; otherwise, the radix is decimal.

Examples

```
.SETN NUMBER 27
```

This directive assigns to the numeric symbol NUMBER the value 27_8.

```
.SETN A1 3*(A2-5)
```

This directive assigns the numeric symbol A1 the value of symbol A2 minus 5, multiplied by 3.

2.6.32 Set Symbol to Octal or Decimal

The .SETO and .SETD directives redefine the radix of a specified numeric symbol. If the symbol has not been defined, Indirect makes an entry in the symbol table and sets the symbol to the specified radix with a value of 0. If the symbol has already been defined, Indirect resets the symbol accordingly. Indirect exits with a fatal error if the symbol was previously defined as a logical or string symbol.

Formats

.SETO ssssss
.SETD ssssss

Parameter

ssssss

The 1- to 6-character numeric symbol to be assigned an octal or decimal radix.

Example

```
.SETN A 10      ; Sets symbol A to 10(8)
.SETD A         ; Defines A as a decimal-radix symbol with a value of
                ; 8(10).
.SETO A         ; Defines A back to original radix with a value of
                ; 10(8).
```

.SETS

2.6.33 Set Symbol to String Value

The .SETS directive defines or changes the string value of a specified string symbol. If the symbol has not been defined, Indirect makes an entry in the symbol table and sets the symbol to the specified string value. If the symbol has been defined, Indirect resets the symbol accordingly. Indirect exits with a fatal error if the symbol was defined previously as a logical or numeric symbol.

Format

.SETS ssssss strexp

Parameters

ssssss

The 1- to 6-character string symbol.

strexp

Any string expression. (See Section 2.4.3.)

Indirect assigns to the specified symbol the string value represented by the string expression strexp. If a string constant is used in strexp, the constant must be enclosed by quotation marks ("constant") or number signs (#constant#).

You can combine a string symbol, constant, or substring with another string symbol or substring by the string concatenation operator (+) to form a string expression.

Examples

```
.SETS A "ABCDEF"
```

This directive assigns to string symbol A the string value ABCDEF.

```
.SETS STR2 "ZZZ"
```

This directive assigns to string symbol STR2 the value ZZZ.

```
.SETS S1 #123"456#
```

This directive assigns to string symbol S1 the value 123"456.

```
.SETS X STR2+"ABC"
```

This directive assigns to string symbol X the value of symbol STR2 plus ABC (that is, ZZZABC).

```
.SETS X STR2+A[1:3]
```

This directive is equivalent to the previous directive. It assigns to string symbol X the string value of STR2 plus the first three characters of string A (that is, ZZZABC).

```
.SETS MYFILE <UIC>+"MYFILE.TXT"
```

This directive assigns the string symbol MYFILE the string value of the current directory and the string contained within the quotation marks (for example, if the current directory is [303,23], MYFILE is assigned the string value [303,23]MYFILE.TXT).

2.6.34 Terminate Command File Processing

The .STOP directive immediately terminates command file processing at all levels, closes all open data files, and exits. The following message is then displayed (unless .DISABLE DISPLAY is in effect):

```
@ <EOF>
```

The .STOP directive allows you to optionally set the exit status for Indirect execution.

Format (brackets not part of syntax)

.STOP [value]

Parameter

value

An optional numeric expression to serve as the exit status for Indirect. If you do not specify an exit status value, the .STOP directive is identical to the logical end-of-file directive (/).

Example

```
.STOP <WARNIN>
```

This directive terminates command file processing and sets the exit status for Indirect to 0 (Warning).

.TEST

2.6.35 Test Symbol

The .TEST directive has two different functions. It tests a variable and sets various special symbols accordingly, and it does substring searches and sets the special symbol <STRLEN> accordingly.

Format 1

.TEST ssssss

Parameter

ssssss
The 1- to 6-character symbol to be tested.

The results of the test are as follows:

- If variable is a string, <SYMTYP> is set to 4 and <STRLEN> contains the length of the string. Also, the special symbols <ALPHAN>, <NUMBER>, <RAD50>, and <OCTAL> are set based on a scan of the characters of variable.
- If variable is numeric, <SYMTYP> is set to 2.
- If variable is octal, <SYMTYP> is set to 2 and <OCTAL> is set to TRUE.
- If variable is logical, <SYMTYP> is set to 0.

Format 2

.TEST string substring

Parameters

string
A string symbol or constant.

substring
A string expression.

In this case, the substring is searched for in the specified string. If the substring is present, <STRLEN> is set to the position of the starting character of the substring within the string. If substring is not present, <STRLEN> is set to 0.

If a string constant is used in string or substring, the constant must be enclosed by quotation marks ("constant") or number signs (#constant#).

Examples

If SUM is a string symbol, the directive statement

```
.TEST SUM
```

sets <SYMTYP> to 4 and places the number of characters represented by the symbol SUM into <STRLEN>.

The directive statements

```
.SETS MAIN "ABCDEF"
.TEST MAIN "C"
```

set <STRLEN> to 3, the position of C in the string ABCDEF.

The directive statements

```
.SETS S1 #AB"CDE#
.TEST S1 #D#
```

set <STRLEN> to 5, the position of D in the string AB"CDE.

.TESTDEVICE

2.6.36 Test Device

The .TESTDEVICE directive allows a command file to acquire information about any device in the system. The information, including error indications, is contained in the string symbol <EXSTRI> . Each device attribute in the string is separated by a comma (which allows processing by the .PARSE and .TEST directives). The first field of the string is the full physical name of the device. The next four fields are octal representations of the device-characteristics words (U.CW1 to U.CW4 of the Unit Control Block). Additional fields contain more information about the device.

Format

.TESTDEVICE dd[nn]:

Parameter

dd[nn]:

The device about which the command file is requesting information.

The device name can also be a logical name assignment that translates into a valid device specification.

The information stored in <EXSTRI> is in the following form:

ddnn:,xx,xx,xx,xx,atr,atr...,atr,

ddnn:

The physical device name for the device specified in the command line.

xx,xx,xx,xx

The four device-characteristics words in octal notation. (See the *RSX-11M-PLUS and Micro/RSX Guide to Writing an I/O Driver* for more information.)

atr

One or more of the following device attributes:

NSD "No such device" is configured into this system.

LOD The device driver is loaded.

UNL The device driver is not loaded.

ONL The device is on line.

OFL The device is off line.

MTD The device is a mountable volume and is mounted.

NMT The device is not a mountable volume or is not mounted.

FOR The device is a mountable volume and is mounted foreign.

NFO The device is not a mountable volume or is not mounted foreign.

PUB The device is a public device.

NPU The device is not a public device.

ATT The device is attached to another task.

ATU The device is attached to this copy of Indirect.

NAT The device is not attached.

ALO The device is allocated to another terminal.

ALU The device is allocated to this terminal.

NAL The device is not allocated.

<EXSTRI> contains the value "NSD," (No such device) if the device is not present in the current system configuration.

<EXSTRI> contains the comma delimiters along with the device information so that the device information can be extracted from <EXSTRI> with the .PARSE directive.

Examples

```
.TESTDEVICE SY:
```

This directive statement acquires information about user logical device SY: and stores it in <EXSTRI> .

```
.TESTDEVICE TEMP
```

This directive acquires information about the device specified by the logical translation of TEMP and stores it in <EXSTRI> .

The command file PROCESS.CMD contains the following directive statement:

```
.TESTDEVICE DATA$DISK
```

When the following DCL command lines are executed:

```
$ DEFINE DATA$DISK DB3: RET
$ @PROCESS RET
```

Indirect acquires information about the device DB3 and stores it in <EXSTRI> .

.TESTFILE

2.6.37 Test a File

The .TESTFILE directive determines if a specified file exists or it determines the physical device associated with a logical name (that is, it performs device translation).

If you specify a file in the command line, the results of a .TESTFILE operation are contained in the symbols <FILSPC> and <FILERR>. <FILSPC> contains the file specification (including device, directory, file name, file type, and version number) and <FILERR> contains the FCS status code resulting from the search for the file.

If you do not specify a file in the command line, Indirect performs device translation only.

Formats

.TESTFILE filespec
.TESTFILE ll:

Parameters

filespec

The file to be tested.

The parameter filespec can also be a logical name assignment that translates into a vaild FCS file specification.

ll:

The logical name assigned to be translated to a physical device.

Examples

```
.TESTFILE MP:IND.MAP
```

If the file exists, this directive assigns the following values:

<FILERR> = 1

<FILSPC> = DU1:[101,300]IND.MAP;4 (MP: is the logical name assigned to the physical device DU1:)

If the file does not exist, the directive assigns the following values:

<FILERR> = 346 (230 decimal)

<FILSPC> = DU1:[101,300]IND.MAP;0

The following directive translates the logical name TI: into its physical device name:

```
.TESTFILE TI:
```

The directive assigns the symbol values as follows:

<FILERR> = 1

<FILSPC> = TT23:.DAT;0

```
.TESTFILE TEMP
```

This directive assigns the symbol values as follows:

<FILERR> = 1

<FILSPC> = File specified by the logical translation of TEMP

The command file SWAN.CMD contains the following directive statement:

```
.TESTFILE EGG
.TESTFILE SYS$TALES
```

When the following DCL command lines are executed:

```
$ ASSIGN SYS$TALES DU2:[UGLY]DUCKLING.DAT [RET]
$ ASSIGN EGG TALES.CMD [RET]
$ @SWAN [RET]
```

Indirect tests for the file DU2:[UGLY]DUCKLING.DAT, and then for the file TALES.CMD in the current directory on the default device.

.TESTPARTITION

2.6.38 Test a Partition

The .TESTPARTITION directive allows a command file to acquire information about a partition in the system. The partition can be the one in which Indirect is executing or any other partition. You can use the directive to verify that a partition is large enough before installing a task in it or that the partition is present before loading a special system. Indirect returns the information (in the special symbol <EXSTRI>) in the following format:

partitionname,base,size,type,

where base and size are in 64-byte blocks and type is SYS for system-controlled partitions, USR for user-controlled partitions, or NSP for an unknown partition name. If the partition is not found, Indirect returns a "No such partition" error in the form:

partitionname,,,NSP,

Format

.TESTPARTITION partitionname

Parameter

partitionname

A 1- to 6-character valid partition name. If you use the wildcard (*) instead of a partition name, Indirect assumes you are testing the same partition in which the current version of Indirect is executing.

Example

```
.TESTPARTITION GEN
;GEN,1500,2303,SYS,
```

This directive acquires information about the partition named GEN. The partition has a starting address of 150000_8, it is 230300_8 bytes long, and it is a system-controlled partition.

2.6.39 Translate a Logical Name Assignment

The .TRANSLATE directive allows a command file to expand a local or global logical name assignment. The expanded assignment is contained in the string <EXSTRI> .

Format

.TRANSLATE ([num]) logical

([num])

A numeric expression, evaluating to a number from 1 to 10_{10}, to specify the number of times to iteratively translate the original logical name assignment, or a wildcard (*) to specify that the assignment should be translated iteratively as many times as possible. If [num] is omitted, it defaults to 1.

logical

The logical name assignment to be expanded.

<EXSTRI> contains a null string if no assignment exists for the specified logical name or if your system does not have support for logical names. If the optional [num] parameter exceeds the number of translation iterations performed, <EXSTRI> contains the results of the final translation.

The special symbol <EOF> is set to true if the expanded logical name assignment is the result of the final iterative translation of the assignment, or if no assignment exists for the specified logical name, or if your system does not have support for logical names. <EOF> is always set to true if you use the wildcard parameter ([*]).

For more information on logical names, especially logical names containing colons, see the descriptions of the ASN and DFL commands in the *RSX-11M-PLUS MCR Operations Manual* or the descriptions of the ASSIGN and DEFINE commands in the *RSX-11M-PLUS Command Language Manual.*

Examples

```
.TRANSLATE TEMP
```

where TEMP is assigned the string "DU0:[USER]LOGIN.CMD"

This directive assigns to the symbol <EXSTRI> the value "DU0:[USER]LOGIN.CMD." <EOF> is set to true if this value is the result of the final iterative translation of the logical name assignment.

```
.TRANSLATE SYS$LOGIN
```

where SYS$LOGIN is assigned the string "DB0:[MYDIR]"

This directive assigns to the symbol <EXSTRI> the value "DU2:[MYDIR]." <EOF> is set to true because SYS$LOGIN is the final iterative translation of the logical name assignment.

.TRANSLATE

```
DFL A=B
DFL B=C
DFL C=D
.TRANSLATE [*] D
.TRANSLATE D
```

The first .TRANSLATE directive assigns to <EXSTRI> the value "A" and <EOF> is set to true. The second .TRANSLATE directive assigns to <EXSTRI> the value "C" and <EOF> is set to false.

2.6.40 Wait for a Task to Finish Execution

The .WAIT directive suspends processing of an indirect command file until a particular task has terminated.

Format

.WAIT taskname

Parameter

taskname

A 1- to 6-character valid task name.

If the task name is omitted, Indirect assumes the task name applied by the last "RUN task" command. This name is specified as follows:

TTnn

TT

The invoking terminal.

nn

The terminal number.

The .WAIT directive also sets the symbol <EXSTAT> with the exit status of the completed task.

If the specified (or default) task is not installed, Indirect ignores the .WAIT directive. The .WAIT directive performs no function if the /NOCLI or /NOMC switch is in effect.

Example

```
.WAIT RUN
```

This directive discontinues processing of the command file until the terminal-initiated task RUN exits.

.XQT

2.6.41 Initiate Parallel Task Execution

Indirect usually passes a command to the CLI and waits until the command's execution has completed. However, it is possible for Indirect to initiate a task and not wait for it to complete before executing the next directive. The .XQT directive allows you to start a task, to pass a command line to it, and to continue processing in parallel with the initiated task. (You can also use the MCR command RUN /CMD="command-line" or the DCL command RUN /COMMAND:"command-line" to pass command lines to another task.) The maximum number of successive .XQT directives allowed depends on the parameter specified in the build file for the Indirect task.

Format

.XQT taskname commandline

Parameters

taskname

The name of the task (for example, MAC or TKB).

commandline

The command line to be executed.

The .XQT directive allows you to initiate parallel processing of tasks. The .WAIT directive is used to synchronize their execution.

The .WAIT directive must also be used to clear out the status block of a .XQT. If the block is not cleared out, it is never reused, which could possibly (after enough .XQTs) produce the error message, "Too many concurrent .XQTs."

If you use the .WAIT directive to clear out the status block, however, you must make sure that the task-name field of the .WAIT directive matches the first three characters of the task-name field of the .XQT directive.

The use of contiguous .XQT/.WAIT directive pairs can also result in a timing problem if you are not executing from MCR. Execution of the .XQT directive from DCL, for example, involves the following sequence of actions:

1. Task spawns DCL to translate command line and pass it to MCR.
2. MCR processes command line and spawns task.
3. Task executes.

Execution of Indirect proceeds as soon as DCL is spawned. Thus, it is possible for the ensuing .WAIT directive to execute before the specified task has been spawned. If this occurs, the .WAIT directive will be ignored, there will be no suspension of Indirect processing, and the status block will not be cleared out. To avoid this problem, use MCR as the CLI.

A second problem could occur if .XQT is used to run a task, as in the following command:

```
.XQT RUN FOO
```

A subsequent .WAIT directive cannot specify FOO in its task-name parameter because only RUN is valid. The .WAIT directive would proceed as soon as FOO started execution (as soon as RUN completed).

This problem can be avoided by using the following sequence of commands:

```
INS FOO/TASK=...DRY
.XQT DRY
.WAIT DRY
```

2.7 Examples

The following sections contain examples showing different uses for Indirect. The longer examples are followed by detailed explanations.

2.7.1 Invoking Indirect Interactively and Displaying Symbols

```
>@TI: RET
AT.>
```

Specifying @TI: allows you to work with Indirect interactively. When Indirect issues the AT.> prompt, you can enter directive statements, invoke command files, or display the values of special symbols. To display a symbol, use the .ENABLE SUBSTITUTION directive, and then request the symbol in the following format:

AT.> ;' <symbol> '

2.7.2 Using an Indirect Command File

A file named PRINTER.CMD contains the following command lines:

```
.ENABLE SUBSTITUTION
;'<TIME>'
PRINT LISTINGS.MEM
.EXIT
```

To execute the command file, use the following command line:

```
>@PRINTER RET
```

2.7.3 Asking for a Device Specification

```
.;
.; This command file asks for a device specification.
.; You may enter the device name with or without a colon
.; and the unit number does not have to be entered for
.; unit 0.  The output produced is the proper device name
.; with a unit number and a colon.
.ENABLE SUBSTITUTION                                                    ❶
.DISABLE LOWERCASE                                                      ❷
.ASKS DEVICE What is the device name?                                   ❸
.SETN TEMPN 2                                                           ❹
.SETS TEMPS ":"                                                         ❺
.TEST DEVICE                                                            ❻
.IF TEMPN EQ <STRLEN> .SETS DEVICE DEVICE+"0"                           ❼
.IF TEMPS NE DEVICE[<STRLEN>:<STRLEN>] .SETS DEVICE DEVICE+":"          ❽
.DISABLE DISPLAY                                                        ❾
; The full device specification is 'DEVICE'                             ❿
.ENABLE DISPLAY                                                         ⓫
.EXIT                                                                   ⓬
```

When you execute this command file, Indirect asks for the name of a device and then displays the complete device specification on the terminal. For example:

```
>@DEVICE [RET]
>* What is the device name? [S:]: du1 [RET]
 The full device specification is DU1:
>@ <EOF>
>
```

The following commentary gives a line-by-line explanation of the command file:

❶ Substitution mode enabled.

❷ Lowercase mode disabled, which means that all input characters are converted to uppercase regardless of how they were typed in.

❸ Asks for the device name (that is, the mnemonic and unit number) and assigns it to the string symbol DEVICE.

❹ Sets numeric symbol TEMPN to the value 2, which is the number of characters for the device mnemonic.

❺ Sets string symbol TEMPS to contain a colon. The colon is a string constant, so it must be enclosed in quotation marks.

❻ Tests the symbol DEVICE (which contains the specified device name). As a result, the following special symbols are set:

<SYMTYP> = 4 (because DEVICE is a string symbol)

<STRLEN> = The length of the string (the number of characters typed in response to the question)

❼ Performs a conditional test. If the value of TEMPN (2) equals the value of <STRLEN>, set DEVICE to be the current contents of DEVICE plus 0. That is, if <STRLEN> equals 2, that means the user typed in the device mnemonic without a unit number. Therefore, the unit number of the device should be 0. DEVICE becomes dd0.

❽ Performs another conditional test. If the value of TEMPS (:) does not equal the last character of DEVICE, add a colon to DEVICE (set the string symbol DEVICE to be equal to DEVICE plus colon; DEVICE becomes ddn:).

❾ Display mode disabled, which means that Indirect displays only the text string for the .ASKS directive and suppresses @ <EOF> upon exiting.

❿ Displays this text, with the full device name substituted for 'DEVICE,' on the terminal.

⓫ Display mode reenabled, which means that @ <EOF> will be displayed after all when Indirect exits.

⓬ Exits from the file and Indirect.

2.7.4 Asking for the Type and Unit Number of the Terminal

```
.ENABLE SUBSTITUTION                    ❶
.ENABLE GLOBAL                          ❷
.TESTFILE TI:                           ❸
.SETN $TRM <TITYPE>                     ❹
; '$TRM'                                ❺
.PARSE <FILSPC> ":" $TT JUNK            ❻
.SETS $TT UNIT                          ❼
; '$TT'                                 ❽
```

When you execute this command file, Indirect retrieves the type and unit number of the terminal from which the file is executing, and displays this information on the terminal. For example:

```
>@TERMINAL RET
>; 15
>; TT14
>@ <EOF>
>
```

The first number in this display, 15, means that the terminal from which the file is running is a VT100. The second number in the display, 14, is the unit number for the terminal.

The following commentary gives a line-by-line explanation of the command file:

❶ Substitution mode enabled.

❷ Global symbol mode enabled, which means that symbol names that begin with a dollar sign ($) are defined as global for all levels of command files. Once such a symbol has been defined, all levels recognize it.

❸ Translates logical name TI: into its physical device name (for example, TT14:) and puts the name in the special symbol <FILSPC>. The device name is turned into a file specification, so the contents of <FILSPC> are TT14:.DAT;0.

❹ Sets numeric global symbol $TRM to the contents of special symbol <TITYPE>, which contains the octal code number for the type of terminal (for example, 15 for a VT100).

❺ Displays the value of $TRM (the octal code number for the terminal type) on the terminal.

❻ Separates the information in the special symbol <FILSPC> into the terminal number (up to, but not including, the colon). The local symbol JUNK contains the characters left over from the special symbol <FILSPC>.

❼ Displays the value of $TT (the unit number of the terminal).

2.7.5 Initializing and Mounting a Volume, and Copying Files to That Volume

In this example, DCL is the CLI for the terminal.

```
        .ENABLE SUBSTITUTION                                    ❶
.GETDEV:                                                        ❷
        .ASKS DEVICE Enter device (DU1 or DU2)                  ❸
        .IF DEVICE EQ "DUO" .GOTO GETDEV                        ❹
        .ASKS DIR What directory (include square brackets)?     ❺
.INIT:                                                          ❻
        .ASK INIT Initialize device                             ❼
        .IFF INIT .GOTO COPY                                    ❽
        ALLOCATE 'DEVICE':                                      ❾
        MOUNT/FOREIGN 'DEVICE'                                  ❿
        .ASKS LABEL What volume label?                          ⓫
        INITIALIZE 'DEVICE':'LABEL'                             ⓬
        DISMOUNT/NOUNLOAD 'DEVICE':                             ⓭
        MOUNT/NOSHAREABLE 'DEVICE':'LABEL'                      ⓮
        CREATE/DIRECTORY 'DEVICE':'DIR'                         ⓯
.COPY:                                                          ⓰
        .ASKS FILES Enter names of files (file1,file2,...)      ⓱
        COPY 'FILES' 'DEVICE':'DIR'                             ⓲
        .ASK MORE More files                                    ⓳
        .IFT MORE .GOTO COPY                                    ⓴
        .ASK LIST List directory                                ㉑
        .IFF LIST .GOTO END                                     ㉒
        DIRECTORY 'DEVICE':'DIR'                                ㉓
.END:                                                           ㉔
        DISMOUNT 'DEVICE':                                      ㉕
        DEALLOCATE 'DEVICE':                                    ㉖
        .EXIT                                                   ㉗
```

The following commentary gives a line-by-line explanation of the command file:

❶ Substitution mode enabled.

❷ Line for .GETDEV: label. It is a direct-access label, so it is the only element on the command line.

❸ Asks for the name of the device to which the files will be copied.

❹ Performs a conditional test. If DEVICE = DU0 (an invalid device), returns to .GETDEV: and asks the question again.

❺ Asks for the directory to which the files will be copied.

❻ Line for the .INIT: label (also a direct-access label).

❼ Asks if the device should be initialized.

❽ If the device should not be initialized, proceeds with the copy operation.

❾ Allocates the specified device.

❿ Mounts the device foreign, which is necessary for initializing a device.

⓫ Asks for the label for the volume.

⓬ Initializes the volume and gives it the specified label.

⓭ Dismounts the device without spinning it down.

⓮ Remounts the device as a private, Files-11 volume.

⓯ Creates the specified directory on the volume.

⓰ Line for the .COPY: label (also a direct-access label).

⓱ Asks for the specifications of the files to be copied.

⓲ Copies the files to the device.

⓳ Asks if there are more files to be copied.

⓴ If there are more files, returns to the .COPY: label.

㉑ If there are no more files, asks if you would like a directory of the copied files.

㉒ If you do not want a directory, goes to the end of the file (.END:).

㉓ If you do want a directory, displays the names of the copied files on the terminal.

㉔ Line for the .END: label (also a direct-access label).

㉕ Dismounts the device.

㉖ Deallocates the device.

㉗ Exits from the file and Indirect.

2.7.6 Editing, Purging, Printing, and Formatting Files

In this example, DCL is the CLI for the terminal.

```
.ENABLE QUIET                                          ❶
.ENABLE SUBSTITUTION                                   ❷

.ASKS FILNAM What is the file name?                    ❸
.ASKS FILTYP What is the file type?                    ❹
EDIT 'FILNAM'.'FILTYP'                                 ❺

.ASK A Do you want to purge this file?                 ❻
.IFT A PURGE/KEEP:2 'FILNAM'.'FILTYP'                  ❼
SET FILE /TRUNCATE 'FILNAM'.'FILTYP';*                 ❽

.ASK DSR Do you want to invoke DSR?                    ❾
.IFT DSR .GOSUB PROC                                   ❿

.ASK B Do you want a listing?                          ⓫
.IFF B .GOTO 100                                       ⓬
.GOSUB LIST                                            ⓭
PRINT/FORMS:'C'/COPIES:'D' 'FILNAM'.'FILTYP'           ⓮
```

```
.100:                                                    ⓯
        .EXIT                                            ⓰
.PROC:                                                   ⓱
       DSR 'FILNAM'='FILNAM'                             ⓲
       .SETS FILTYP "MEM"                                ⓳
       .ASK F Do you want to purge the .MEM files?       ⓴
       .IFT F PURGE/KEEP:2 'FILNAM'.MEM                  ㉑
       SET FILE /TRUNCATE 'FILNAM'.MEM;*                 ㉒
       .RETURN                                           ㉓
.LIST:                                                   ㉔
       .ASKN C What form number?                         ㉕
       .ASKN [::1.] D How many copies?                   ㉖
       .RETURN                                           ㉗
```

The following commentary gives a line-by-line explanation of the command file:

❶ Quiet mode enabled, which means that Indirect does not echo (display on the terminal) DCL command lines or comments. The command lines are executed normally and, if they return a message or display, those are shown on the terminal.

❷ Substitution mode enabled.

❸ Asks for the name of the file (for example, MYFILE).

❹ Asks for the type of the file (for example, CMD).

❺ Invokes EDT so that you can edit the specified file.

❻ When you are done with EDT (using the EXIT or QUIT command), asks if you want to purge the versions of the file.

❼ If you want to purge the files, DCL does so, keeping the two latest versions of the file.

❽ Truncates the files to free up blocks that are allocated to the files but not used.

❾ Asks if you want to use DIGITAL Standard Runoff (DSR) to format the file.

❿ If you do want to use DSR, Indirect goes to the subroutine for file processing (.PROC:).

⓫ After returning from the processing subroutine, asks if you want a listing of the file.

⓬ If you do not want a listing, exits from the file and Indirect.

⓭ If you do want a listing, goes to the subroutine for listing files (.LIST:).

⓮ After returning from the listing subroutine, prints the specified number of copies of the file on the designated printer.

⓯ Line for label .100: (a direct-access label).

⓰ Exits from the file and Indirect.

⓱ Line for .PROC: label (label for the processing subroutine).

⓲ DSR formats the file (which must be a RNO file) and creates (by default) a MEM file.

⓳ Sets string symbol FILTYP equal to type MEM.

⓴ Asks if you want to purge the MEM files.

㉑ If you do want to purge the files, DCL does so, keeping the two latest versions of the file.

㉒ Truncates the files to free up blocks that are allocated to the files but not used.

㉓ Returns to the line after .GOSUB PROC (.ASK B Do you want a listing).

㉔ Line for .LIST: label (label for the listing subroutine).

㉕ Asks for the form number for the line printer. Sets the numeric symbol C to this value, which is used in the PRINT command line.

㉖ Asks for the number of copies to be printed (the default is 1). Sets numeric symbol D to this value, which is also used in the PRINT command line.

㉗ Returns to the line after .GOSUB LIST (the PRINT command line).

Chapter 3
The Indirect Pre-Processor (IPP)

This chapter describes the Indirect Pre-Processor (IPP). IPP is a utility that enables you to preprocess indirect command files to improve performance.

The output form of IPP is another command file, which is interchangeable with the original command file with respect to functions performed and the user interface.

In general, IPP performs the following functions:

- Reads an indirect command file and changes all directives and special symbol notations to a shortened form
- Writes a new file, with the type CMF, that has the shortened command lines

An indirect command file that has not been preprocessed requires that all directives and special symbols be spelled out completely in the command lines. As each directive is encountered during execution of the file, Indirect searches an internal jump table to select the appropriate processing routine. Each occurrence of a special symbol also requires Indirect to search a jump table. The search implements the processing of information to be provided for that symbol.

Directive and special symbols in a preprocessed command file, however, have been replaced by a 2-character code that consists of an identifier (! or ~) and an index into the jump tables. Because the index is used to access the appropriate processing routine directly, Indirect does not have to spend time searching the tables.

Replacing the multicharacter directives and symbols with a 2-character code also reduces the size of the command file. This reduction improves performance and minimizes disk-space requirements. When command files are shorter, fewer disk accesses are required to read the files and more command lines can be memory-resident at any one time in the internal block buffers for the command file.

In general, indirect command files that are executed frequently (for example, LOGIN.CMD) benefit the most from preprocessing. Files that are likely to change frequently would not benefit as much. However, the characters used for the codes are all printable so that the preprocessed command files can be edited if necessary. Also, you can edit the original CMD file and preprocess it to incorporate changes.

3.1 Invoking IPP

You can invoke IPP by using any of the methods described in Chapter 1 of the *RSX-11M-PLUS Utilities Manual.* They are summarized as follows:

- Invoking an installed IPP task and returning to your CLI:

```
>IPP command-line
```

- Invoking IPP for interactive use:

```
>IPP
IPP>command-line
  .
  .
IPP> CTRL/Z
```

- Invoking an uninstalled IPP task for interactive use:

```
>RUN $IPP
IPP>command-line
  .
  .
IPP> CTRL/Z
```

- Using indirect command files:

```
>IPP @command-file
```

or

```
>IPP
IPP>@command-file
```

or

```
>RUN $IPP
IPP>@command-file
```

The command line format for IPP is as follows:

Format

IPP> [outfile [.typ]] = infile [.typ]

outfile

Specifies the output file. If the output file name is not specified, IPP uses the file name specified in the infile parameter. If the output file type is not specified, IPP uses the default type CMF.

infile

Specifies the input file. If the input file type is not specified, IPP uses the default type CMD.

When Indirect is invoked by @filename command, and the device, directory, and file type are all unspecified, it will search for the command file in the following sequence:

1. Default device and directory, filename.CMF
2. Default device and directory, filename.CMD
3. Library device and directory, filename.CMF
4. Library device and directory, filename.CMD

3.2 Short Forms

IPP changes all directives and special symbols to a shortened form, and writes a new file incorporating the resulting command lines. Each directive maps to a 2-character notation with the form !x, where x is one of the 94 printable characters from the exclamation point (!) to the tilde (~). Similarly, the special symbols map to the notation ~y, where y is one of the 94 printable characters.

Tables 3–1 and 3–2 show the Indirect directives and special symbols and their corresponding short forms.

Table 3–1: Short Forms for Indirect Directives

Directive	Short Form	Directive	Short Form
.AND	q	.IFNINS	r
.ASK	>	.IFNLOA	w
.ASKN	;	.IFT	l
.ASKS	=	.INC	9
.BEGIN	M	.ONERR	P
.CHAIN	N	.OPEN	I
.CLOSE	@	.OPENA	G
.DATA	4	.OPENR	H
.DEC	F	.OR	p
.DELAY	V	.PARSE	A
.DISABLE	5	.PAUSE	O
.ENABLE	6	.READ	Q
.END	]	.RETURN	<
.ERASE	?	.SETD	R
.EXIT	K	.SETF	7
.FORM	X	.SETL	:
.GOSUB	3	.SETN	2
.GOTO	0	.SETO	L
.IF	z	.SETS	1
.IFACT	u	.SETT	8
.IFDF	o	.STOP	W
.IFDISABLED	y	.TEST	E
.IFENABLED	x	.TESTDEVICE	C
.IFF	m	.TESTFILE	B
.IFINS	s	.TESTPARTITION	D
.IFLOA	v	.TRANSLATE	S
.IFNACT	t	.WAIT	U
.IFNDF	n	.XQT	T

Table 3–2: Short Forms for Indirect Special Symbols

Special Symbol	Short Form	Special Symbol	Short Form
<ACCOUN>	k	<LOGUIC>	e
<ALPHAN>	7	<MAPPED>	0
<ALTMOD>	3	<MEMSIZ>	C
<BASLIN>	=	<NETNOD>	h
<CLI>	^	<NETUIC>	g
<CONFIG>	I	<NOSTAT>	N
<DATE>	_	<NUMBER>	8
<DEFAUL>	5	<NXTSYM>	j
<DIRECT>	b	<OCTAL>	9
<EOF>	;	<PRIVIL>	<
<ERRCTL>	S	<RAD50>	6
<ERRNUM>	Q	<RSX11D>	1
<ERROR>	L	<SEVERE>	M
<ERRSEV>	R	<SPACE>	>
<ERSEEN>	T	<STRLEN>	F
<ESCAPE>	4	<SUCCES>	J
<EXSTAT>	G	<SYDISK>	\
<EXSTRI>	H	<SYMTYP>	E
<FALSE>	P	<SYSDEV>	]
<FILATR>	[	<SYSID>	@
<FILER2>	B	<SYSTEM>	?
<FILERR>	A	<SYSUIC>	c
<FILSPC>	l	<SYTYP>	m
<FMASK>	i	<SYUNIT>	D
<FORATT>	U	<TICLPP>	X
<IAS>	2	<TICWID>	Y
<LIBUIC>	d	<TIME>	`
<LOCAL>	Z	<TIMOUT>	:
<LOGDEV>	f	<TISPED>	W

Table 3–2 (Cont.): Short Forms for Indirect Special Symbols

Special Symbol	Short Form	Special Symbol	Short Form
<TITYPE>	V	<VERSN>	n
<TRUE>	O	<WARNIN>	K
<UIC>	a		

Example

The following example shows a command file before and after it was processed with IPP. This is a file with the file type CMD that you create with an editor:

```
.;
.; This command file asks for a device specification.
.; You may enter the device name with or without a colon
.; and the unit number does not have to be entered for
.; unit 0.  The output produced is the proper device name
.; with a unit number and a colon.
.ENABLE SUBSTITUTION
.DISABLE LOWERCASE
.ASKS DEVICE What is the device name?
.SETN TEMPN 2
.SETS TEMPS ":"                                    ! Line 5
.TEST DEVICE
.IF TEMPN EQ <STRLEN> .SETS DEVICE DEVICE+"0"
.IF TEMPS NE DEVICE[<STRLEN>:<STRLEN>] .SETS DEVICE DEVICE+":"
.DISABLE DISPLAY
; The full device specification is 'DEVICE'      ! Line 10
.ENABLE DISPLAY
.EXIT
```

The following is the same file after it has been processed with IPP. This file has the type CMF:

```
|6 SUBSTITUTION
|5 LOWERCASE
|= DEVICE What is the device name?
|2 TEMPN 2
|1 TEMPS ":"
|E DEVICE
|z TEMPN EQ ^F |1 DEVICE DEVICE+"0"
|z TEMPS NE DEVICE[<STRLEN>:<STRLEN>] |1 DEVICE DEVICE+":"
|5 DISPLAY
; The full device specification is 'DEVICE'      ! Line 10
|6 DISPLAY
|K
```

Note the following differences in the command file after it has been processed with IPP:

- All Indirect directives and special symbols have been replaced by their corresponding short forms.
- All comment lines beginning with a period and a semicolon (.;) have been omitted.

- The ! Line 5 comment has been omitted.
- The ! Line 10 comment has been retained because it appears on a comment line that begins with a semicolon (;), and is displayed on the terminal when the command file is executed.
- The special symbols enclosed within square brackets have not been translated to their short forms. Special symbols and directives that are contained within square brackets are not put into their short forms when a file is preprocessed.

Appendix A
Indirect Messages

When Indirect encounters an error, it displays the appropriate error message and the command line in which the error occurred. If the line contained a substitution, the line as it appeared before the substitution took place is also displayed. Indirect also closes all open data files before exiting.

Section A.1 explains the information-only messages and Section A.2 explains the error messages. The error messages are divided into four classes, depending on the level of severity. Class 2 errors can be handled with the <ERRCTL> symbol (see Section 2.4.1.2) and class 1 errors can be handled with the .ONERR directive (see Section 2.6.22). Class 0 errors must be corrected outside of Indirect. The remaining messages are only for your information.

A.1 Information-Only Messages

@ < EOF>

Explanation: (Class 0) Indirect has reached the end-of-file for the outermost command file and is terminating execution.

AT. — Continuing

Explanation: Indirect is resuming execution after a .PAUSE or .DELAY directive.

AT. — Delaying

Explanation: A .DELAY directive was just executed, halting the processing of an indirect command file for a specified period of time.

AT. — Invalid answer or terminator

Explanation: In response to a question from .ASK, you entered something other than Y, N, or null, followed by a RETURN; or you did not enter a numeric value in response to a .ASKN question; or you pressed the ESCAPE key either without escape recognition enabled or as a character other than the first one following the question. The question will be repeated.

AT. — Pausing. To continue type "command taskname"

Explanation: Indirect just executed a .PAUSE directive, interrupting processing of an indirect command file to wait for user action.

AT. — Value not in range

Explanation: The response to a .ASKN or .ASKS question was not within the specified range. Indirect repeats the question. Or, the time specified for a .DELAY directive exceeded 24 hours.

A.2 Error Messages

AT. — Bad range or default specification

Explanation: (Class 1) An illegal character was specified as a range or default argument. Only numeric expressions are permitted.

AT — Command file open error

Explanation: (Class 2) The file being invoked in an @file or @file/LB:module command line cannot be found or opened.

AT. — Data file error, code x.

Explanation: (Class 1) Indirect encountered an error while processing a .OPEN, .OPENA, .CLOSE, or .DATA directive, or a data-mode access to the secondary file. See the description of <FILERR> (Section 2.4.1.2) for a definition of the numeric code x.

AT. — .EXIT without .END

Explanation: (Class 1) After executing a .EXIT directive from within a Begin-End block, Indirect encountered an end-of-file before finding a .END directive.

AT. — File already open

Explanation: (Class 1) A .OPEN or .OPENA directive specified a file that was already open.

AT. — File attributes not available

Explanation: (Class 1) An attempt was made to obtain file-attribute information with the <FILATR> symbol before any files were opened.

AT. — File not found

Explanation: (Class 2) An @filename or .CHAIN directive specified an incorrect file name or nonexistent file.

AT. — File not open

Explanation: (Class 1) Indirect encountered a .DATA or .CLOSE directive that did not reference an open file.

AT. — File read error

Explanation: (Class 2) An error was detected in reading the indirect command file. This error is usually caused by records that are more than 132_{10} bytes long.

AT. — Illegal file number

Explanation: (Class 1) The file number in a .OPEN, .OPENA, .OPENR, .DATA, .ENABLE DATA, .READ, or .CLOSE directive is not in the range of 0 to 3.

AT. — Illegal nesting

Explanation: (Class 1) Too many Begin-End blocks have been nested in the indirect command file. The maximum nesting depth is limited to the size of the symbol table.

AT. — Initialization error, code x.

Explanation: (Class 0) Indirect failed to complete initialization when you invoked it. The following list gives the meaning of the displayed code number:

1. Unable to acquire system information such as the directory or device name.
2. Impure area setup failed.
3. Unable to acquire task-specific information.
4. Unable to acquire terminal-type information.
5. Unable to acquire the disk name and other information about the system device (SY).
6. Unable to allocate enough space for command and data I/O buffers. For privileged Indirect tasks, Indirect was not installed with a large enough increment value. The system manager should remove and reinstall Indirect with a larger increment or in a larger partition. For the nonprivileged Indirect task, the Executive directive Extend Task failed to return sufficient space for Indirect to allocate the buffers.
7. Initialization of allocated buffers failed.
8. Initialization of the DATA file structures failed.
9. Allocation of FCS-11 buffers for data and command lines failed.
10. Symbol table initialization failed.
11. Initialization cleanup failed.
12. Unable to obtain initial command line.

$>$ 12. Error codes greater than 12 are returned by FMS-11 and other special-purpose initialization modules.

Error number 6 is the only initialization error that you should encounter. If any other error from 1 through 12 persists, submit a Software Performance Report (SPR) with any other pertinent information.

AT. — Invalid keyword

Explanation: (Class 1) An unrecognized keyword (preceded by a period) was specified.

AT. — Label not at beginning of line

Explanation: (Class 1) The specified label does not start in the first column of the line. All labels must do so.

AT. — Logical name translation error

Explanation: (Class 1) A logical name translation directive error has occurred. Use the MCR DFL or DCL SHOW LOGICALS command to determine the status of your logical name assignments.

AT. — Maximum indirect file depth exceeded

Explanation: (Class 2) An attempt was made to reference an indirect command file at a nested depth greater than the maximum specified in the build file for the Indirect task.

AT. — No pool space

Explanation: (Class 2) The dynamic memory allocation has been exhausted. Either wait for more pool space to become available or use the MCR command SET /POOL or the DCL command SET SYSTEM/POOL.

AT. — Null control string

Explanation: (Class 1) The control string specified with the .PARSE directive was null (there were no characters between the quotation marks or number signs).

AT. — Numeric under- or overflow

Explanation: (Class 2) The evaluation of a numeric expression yielded a value outside the range 0 to 177777_8. This means that the value crossed the zero boundary from positive to negative or negative to positive.

AT. — Redefining a read-only symbol.

Explanation: (Class 2) An attempt was made to assign a new value to a read-only symbol. Read-only symbols cannot be overwritten.

AT. — Redefining symbol to different type < ssssss>

Explanation: (Class 1) A .ASK, .ASKN, .ASKS, .READ, .SETT, .SETF, .SETL, .SETN, or .SETS directive was used in an attempt to set the specified, already defined symbol to a different type. The first definition of a symbol determines its type (logical, numeric, or string); subsequent value assignments must conform to the original type.

AT. — .RETURN without .GOSUB

Explanation: (Class 1) A .RETURN directive was specified without a previous call to a subroutine (.GOSUB).

AT. — Spawn failure

Explanation: (Class 1) Indirect could not initiate the execution of a user command task.

AT. — String expression larger than 132. bytes

Explanation: (Class 2) An attempt was made to generate a string expression longer than 132_{10} characters.

AT. — String substitution error

Explanation: (Class 1) Indirect encountered an error during a substitution operation. A probable cause for the error is either the omission of a second apostrophe or the specification of a symbol that is not defined.

AT. — Subroutine nesting too deep

Explanation: (Class 1) The maximum subroutine nesting level was exceeded. The maximum level is specified in the build file for the Indirect task.

AT. — Symbol table overflow < ssssss>

Explanation: (Class 2) The symbol table was full and there was no space for symbol ssssss.

AT. — Symbol type error < ssssss>

Explanation: (Class 1) The symbol ssssss was used out of context for its type; for example, a numeric expression referenced a logical symbol. Only symbols of the same type can be compared.

AT. — Syntax error

Explanation: (Class 2) The format of the specified command line is incorrect.

AT. — Too many concurrent .XQTs

Explanation: (Class 1) More than the maximum number of successive .XQT directives allowed by the build file for the Indirect task were issued.

AT. — Undefined label < .label:>

Explanation: (Class 1) The label .label: specified with a .GOTO, .GOSUB, or .ONERR directive could not be found.

AT. — Undefined symbol < ssssss>

Explanation: (Class 1) The symbol ssssss was referenced, but it had not been defined.

Index

F

G

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

RSX–11M–PLUS
Indirect Command Processor
Manual
AA–JS10A–TC

READER'S COMMENTS

Your comments and suggestions are welcome and will help us in our continuous effort to improve the quality and usefulness of our documentation and software.

Remember, the system includes information that you read on your terminal: help files, error messages, prompts, and so on. Please let us know if you have comments about this information, too.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

What kind of user are you? ___ Programmer ___ Nonprogrammer

Years of experience as a computer programmer/user: _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code _____
or Country

Do Not Tear - Fold Here and Tape

digital

No Postage Necessary if Mailed in the United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
The Manager, Office Program
ZK02-1/N20
110 SPIT BROOK ROAD
NASHUA, NH 03062 - 9990

Do Not Tear - Fold Here

RSX–11M–PLUS
Indirect Command Processor
Manual
AA–JS10A–TC

READER'S COMMENTS

Your comments and suggestions are welcome and will help us in our continuous effort to improve the quality and usefulness of our documentation and software.

Remember, the system includes information that you read on your terminal: help files, error messages, prompts, and so on. Please let us know if you have comments about this information, too.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

What kind of user are you? ___ Programmer ___ Nonprogrammer

Years of experience as a computer programmer/user: _____

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code _____
or Country

— — Do Not Tear - Fold Here and Tape —

digital

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
The Manager, Office Program
ZK02-1/N20
110 SPIT BROOK ROAD
NASHUA, NH 03062 - 9990

— — Do Not Tear - Fold Here —