

# **IAS Indirect Command Processor Manual**

Order Number: AA-PAXUA-TC

**Operating System and Version:** IAS Version 3.4

---

**May 1990**

---

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

Restricted Rights: Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by Digital Equipment Corporation or its affiliated companies.

---


Copyright ©1990 by Digital Equipment Corporation

All Rights Reserved.  
Printed in U.S.A.

---

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:

DDIF	IAS	VAX C
DEC	MASSBUS	VAXcluster
DEC/CMS	PDP	VAXstation
DEC/MMS	PDT	VMS
DECnet	RSTS	VR150/160
DECUS	RSX	VT
DECwindows	ULTRIX	
DECwrite	UNIBUS	
DIBOL	VAX	

This document was prepared using VAX DOCUMENT, Version 1.2

---

# Contents

---

PREFACE	vii
---------	-----

---

<b>CHAPTER 1 INTRODUCTION TO INDIRECT</b>	<b>1-1</b>
---	------------

---

1.1	INDIRECT COMMAND PROCESSING	1-1
1.2	SUBSTITUTION MODE	1-1
1.3	WRITING PROGRAMS WITH INDIRECT	1-2
1.3.1	Directives _____	1-2
1.3.2	Special Symbols _____	1-3
1.3.3	Labels _____	1-3
1.4	EXPLANATION OF THE COMMAND FILE	1-4
1.5	EXAMPLES	1-5

---

<b>CHAPTER 2 THE INDIRECT COMMAND PROCESSOR (REFERENCE SECTION)</b>	<b>2-1</b>
---	------------

---

2.1	INDIRECT COMMAND FILES	2-1
2.1.1	Indirect Task Command Files _____	2-1
2.1.2	Indirect MCR Command Files _____	2-2
2.2	THE INDIRECT COMMAND PROCESSOR	2-2
2.3	SUMMARY OF INDIRECT DIRECTIVES	2-4
2.4	SYMBOLS	2-6
2.4.1	Special Symbols _____	2-7
2.4.1.1	Special Logical Symbols • 2-7	
2.4.1.2	Special Numeric Symbols • 2-8	

2.4.2	Numeric Symbols and Expressions _____	2-9
2.4.3	String Symbols, Substrings, and Expressions _____	2-10
2.4.4	Logical Symbols and Expressions _____	2-11
2.4.5	Reserved Symbols _____	2-12
2.4.6	Symbol Value Substitution _____	2-12
<hr/>		
2.5	SWITCHES	2-13
<hr/>		
2.6	DESCRIPTION OF INDIRECT DIRECTIVES	2-14
	.LABEL:	2-15
	.ASK	2-16
	.ASKN	2-18
	.ASKS	2-21
	.CHAIN	2-23
	.CLOSE	2-24
	.DATA	2-25
	.DEC	2-27
	.DELAY	2-28
	.DISABLE	2-29
	.ENABLE	2-30
	.EXIT	2-33
	.GOSUB	2-34
	.GOTO	2-35
	.IF	2-36
	.IFACT/.IFNACT	2-38
	.IFDEV/.IFNDEV	2-39
	.IFDF/.IFNDF	2-41
	.IFFILE/.IFNFILE	2-42
	.IFINS/.IFNINS	2-44
	.IFLOA/.IFNLOA	2-45
	.IFMOU/.IFNMOU	2-46
	.IFPAR/.IFNPAR	2-47
	.IFT/.IFF	2-48
	.IFREADY/.IFNREADY	2-49
	.INC	2-51
	/	2-52
	.ONERR	2-53
	.ONFAIL	2-56
	.OPEN	2-58
	.OPENA	2-59
	.OPENR	2-60
	.PARSE	2-61
	.PAUSE	2-62
	.READ	2-63
	.RETURN	2-64
	.SEARCH	2-65
	.SETT/.SETF	2-66
	.SETN	2-67
	.SETS	2-68
	.TASK	2-71

.TEST	2-72
.WAIT	2-75
.XQT	2-76

---

<b>2.7</b>	<b>EXAMPLES</b>	<b>2-77</b>
2.7.1	Using an Indirect Command File _____	2-77
2.7.2	Asking for a Device Specification _____	2-77
2.7.3	Initializing and Mounting a Volume, and Copying Files to That Volume _____	2-78
2.7.4	Editing, Purging, Printing, and Formatting Files _____	2-79

---

**APPENDIX A INDIRECT MESSAGES** **A-1**

---

<b>A.1</b>	<b>INFORMATION-ONLY MESSAGES</b>	<b>A-1</b>
------------	----------------------------------	------------

---

<b>A.2</b>	<b>ERROR MESSAGES</b>	<b>A-1</b>
------------	-----------------------	------------

---

**INDEX**

---

**TABLES**

2-1	Indirect Switches _____	2-13
-----	-------------------------	------



---

# Preface

---

## Manual Objectives

The *IAS Indirect Command Processor Manual* describes Indirect, the task used to run indirect MCR 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.

---

## Document Structure

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.

Appendix A lists and explains all of the Indirect messages.

---

## Associated Documents

The *IAS MCR Operations Manual* supplements this manual in the following ways:

- It gives more detail about the way the system operates.
- It describes the commands mentioned in this manual.

---

## Documentation Conventions

---

Convention	Meaning
.	The vertical ellipsis shows where elements of command input have been omitted because they are not relevant to the point being discussed or where elements of command point are 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.

## Preface

Convention	Meaning
[g,m]	<p>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.</p> <p>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].</p>
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	<p>A full file specification includes device, directory, file name, file type, and version number, as in this example:</p> <pre>DL2:[46,63]INDIRECT.TXT;3</pre> <p>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.</p>
<b>RET</b>	A rectangular symbol with a 2- to 6-character abbreviation indicates that you are to press the corresponding key on your terminal. For example, <b>RET</b> indicates that you are to press the RETURN key and <b>DEL</b> means that you are to press the DELETE key.
<b>CTRL/a</b>	The rectangular symbol <b>CTRL/a</b> means that you are to press the key marked CTRL while pressing another key. Thus, <b>CTRL/Z</b> indicates that you are to press the CTRL key and the Z key simultaneously. <b>CTRL/Z</b> is echoed on your terminal as ^Z, but not all control characters echo.



# 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.

---

### 1.1 Indirect Command Processing

You create a file and put the MCR 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. Then Indirect and MCR do all the work.

For example, the command file EXAMPLE.COMD contains the following MCR command lines:

```
TIME
PIP WORKLIST.TSK;*/LI
RUN WORKLIST
QUE WORKLIST.MAP,WORKLIST.LST
TIME
```

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

```
MCR> @EXAMPLE 
```

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, 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.

---

### 1.2 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.

## Introduction to Indirect

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 MCR commands and for questions displayed by the `.ASKx` directives, the first character displayed is the right angle bracket (`>`).

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.

---

## 1.3 Writing Programs with Indirect

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.

---

### 1.3.1 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.

### 1.3.2 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.

### 1.3.3 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. 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.
```

## Introduction to Indirect

```
.000:      ; What else can go wrong?
          .GOTO END
.100:      ; Bicycles don't have doors.
          .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
```

---

### 1.4 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 (.;) do not display on the terminal when the file is executed. Comments that begin with only a semicolon (;) display.

This file, as the introductory comment explains, displays a message on the terminal when the file is run. The message displayed depends on when 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 branches 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 g

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.

## 1.5 Examples

An explanation of the example follows each one.

- The following MCR 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 'DISK'/FOR/ATCH
BAD 'DISK'
INITIALIZE 'DISK' 'LABEL'
DMO 'DISK'
MOUNT 'DISK' 'LABEL'
; 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 IAS system.

The `BAD` 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.

## Introduction to Indirect

The INITIALIZE command writes a new file structure on the diskette so that the diskette is in FILES-11 format. FILES-11 is the standard IAS format for disk volumes. The DMO 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 last line of the command file displays the statement that the specified diskette is now ready for you to use.

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

```
.ENABLE SUBSTITUTION
.BEGIN:
.ASKS FILE Which file?
PIP TI: = 'FILE'
.ASK DEL Delete this file
.IFT DEL PIP '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, PIP 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, PIP 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.
'<SYDISK>'<SYUNIT>' ! This is your login device.
.DISABLE DATA
.CLOSE INFO.DAT
PIP TI: = 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:

```
MCR> @INFORM RET
MCR> PIP TI: = 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.
MCR> @ <EOF>
MCR>
```

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

**NOTE:** 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.





---

## 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.

---

### 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.

Indirect command files are of two different types:

- Indirect task command files
- Indirect MCR command files

Sections 2.1.1 and 2.1.2 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 the IAS operating system, 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:

```
MCR>MAC @INPUT.CMD 
```

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:

```
MCR>MAC @INPUT 
```

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

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 MCR Command Files

An indirect MCR command file is a text file containing MCR command lines and special directives that enable 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 MCR commands to MCR.

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

```
.ENABLE SUBSTITUTION
.ASKS COMMAN Enter file to delete
PIP 'COMMAN';0/DE/LD
```

With this file, Indirect processes the first two command lines and PIP does the following:

- Deletes the current version of the specified file
- Displays its action

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

```
MCR> @COMMANDS.CMD [RET]
```

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

```
MCR> @COMMANDS [RET]
```

Indirect MCR command files can also be nested. To illustrate, a 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 an MCR 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 MCR commands.

```
TER TI:/VT100
@COOKIE
DEV
DEV/LOG
TIME
```

For MCR commands and for questions displayed by the .ASKx directives, the first character displayed is the right angle bracket (>).

The Indirect directives described in Section 2.6 can be used in indirect MCR command files. All further references in this chapter to indirect command files apply to indirect MCR 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 MCR or as a request for action to Indirect. The directives for Indirect are distinguished by a period (.) as their beginning character.

The Indirect directives enable 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.

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).

One Indirect directive, `.ENABLE GLOBAL` (see the `.ENABLE` directive), 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.)

Indirect displays on the requesting terminal every MCR command line as it is executed. However, if Indirect is activated by `@filename/NOMCR`, the MCR command lines are displayed but not executed. (See Section 2.5 for information on the `/[NO]MCR` 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 MCR to treat them. Following are the three formats for comments:

<code>;</code> comment	Comments at beginning of line to be displayed by the MCR
<code>!</code> comment	Comments after the start of a MCR command line
<code>.;</code> comment	Comments that are not displayed

**NOTE: Command and comment lines are not displayed if `.ENABLE QUIET` is in effect.**

## The Indirect Command Processor (Reference Section)

References to task names in an indirect command file follow the rules used for MCR. If the task was started as an external MCR task (for example, MAC, PIP, DMO), it can be referenced by its full 6-character name (...xxx or \$\$\$xxx).

### 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
<b>Label Definition</b>	
.label:	Assigns a name to a line in the command file so the line can be referenced elsewhere within the file by a .GOTO or .GOSUB directive.
<b>Symbol Definition</b>	
.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 numeric value.
.ASKS	Prompts for user input to define or redefine a string symbol and assign the symbol a character string value.
.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.
.SETS	Defines or redefines a string symbol and assigns the symbol a character string value.
<b>File Access</b>	
.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.
.SEARCH	Determines the position of a substring within a given string.
.TASK	Opens a task image file and reads task image size from label block.

## The Indirect Command Processor (Reference Section)

Category	Function
<b>Logical Control</b>	
.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.
.ONFAIL	Branches to a label upon detecting an error return status from MCR commands.
.RETURN	Effects an exit from a subroutine and returns to the line immediately following the subroutine call.
<b>Logical Tests</b>	
.IF	Determines whether or not a symbol satisfies a condition.
.IFACT	Determines whether or not a task is active.
.IFNACT	
.IFDEV	Tests whether or not a PUD exists for a device in the system.
.IFNDEV	
.IFDF	Determines whether or not a symbol is defined.
.IFNDF	
.IFFILE	Determines whether or not a file exists.
.IFNFILE	
.IFINS	Determines whether or not a task is installed in the system.
.IFNINS	
.IFLOA	Determines whether or not a device handler is loaded.
.IFNLOA	
.IFMOU	Tests whether or not a device is mounted.
.IFNMOU	
.IFREADY	Determines whether or not a disk is online.
.IFNREADY	
.IFT	Determines whether a logical symbol is true or false.
.IFF	
.TEST	Tests the length of a string symbol or locates a substring.

## The Indirect Command Processor (Reference Section)

Category	Function
<b>Enable or Disable an Operating Mode</b>	
.ENABLE .DISABLE	Enables or disables control of the following modes: Substitution (SUBSTITUTION) Statistics (STATISTICS) Output of data to data files (DATA) Global symbols (GLOBAL) Symbol radix (DECIMAL) Command line echo (QUIET) Field display (DISPLAY) Case sensitivity (LOWERCASE) Passing commands to MCR (MCR) Command-display (LIST) Collecting direct-access labels in a table (LRU) Providing time stamp with MCR> prompts (TIME) Escape recognition (ESCAPE)
<b>Increment or Decrement Numeric Symbols</b>	
.DEC .INC	Decrements the value of a numeric symbol by 1. Increments the value of a numeric symbol by 1.
<b>Execution Control</b>	
.DELAY .PAUSE .WAIT .XQT	Delays the execution of an indirect command file for a specified period of time. Temporarily suspends the execution of an indirect command file to enable user action. Waits for a specified task to complete execution and sets the special symbol EXSTAT with the completed task's exit status. Initiates a task, passes a command line to it, and continues Indirect processing without waiting for the task to complete.

## 2.4 Symbols

Indirect enables you to define symbols. These symbols can then be tested or compared to control flow through the indirect command file. Their values can also be inserted into MCR 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<sub>8</sub> (65,535<sub>10</sub>). 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<sub>10</sub> characters in length.

A symbol 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`, `.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 can 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 Section 2.4.6 discusses 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 <code>.ASKS</code> directive or tested with a <code>.TEST</code> directive contains only alphanumeric characters. An empty string also sets ALPHAN to true.
<ALTMOD>	Set to true if last question was answered with an <code>ALTMODE</code> or <code>ESCAPE</code> . Otherwise, ALTMOD is set to false.
<DEFAULT>	Set to true if the answer to the last query was defaulted (the <code>RETURN</code> key was pressed once) or a timeout occurred.
<EOF>	Set to true if the last <code>.READ</code> or <code>.ASKx</code> 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 <code>.TRANSLATE</code> directive resulted in a final logical translation assignment.
<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 <code>.IF</code> directive or as a default for the <code>.ASK</code> directive.
<IAS>	Always TRUE on IAS systems.
<MAPPED>	Always true on IAS systems.
<OCTAL>	Set to true if the answer to the last <code>.ASKN</code> directive or the radix of the numeric symbol tested in the last <code>.TEST</code> directive is octal, or if the last string tested with a <code>.TEST</code> directive contained all numeric characters in the range 0 to 7.

## The Indirect Command Processor (Reference Section)

Symbol	Value
<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>	Always TRUE on IAS systems.
<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
<ERROR>	Value of the exit status code returned by a task that has issued error messages.
<EXSTAT>	Assigned the value of 0, 1, 2, 4, or 17, depending on the exit status from the last MCR 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 MCR 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 MCR. The values 0, 1, 2, 4, and 17 and their corresponding special symbols indicate: 0-WARNIN      Warning 1-SUCCESS    Success 2-ERROR       Error 4-SEVERE      Severe error 17-NOSTAT     The task could not return exit status.
<MEMSIZ>	Assigned the value of the current system memory size in K words (K is 1024 <sub>10</sub> ).
<SEVERE>	Value of the exit status code returned by a task that has issued error messages.
<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 . )
<SUCCESS>	Value of the exit status code returned by a successfully executed task.
<SYSTEM>	Assigned an octal number to represent the operating system on which Indirect is running. For an IAS system, the value is always 3.
<SYUNIT>	Assigned the unit number of the user's default device (SY).
<TSKTSZ>	Value returned by the .TASK directive that represents the size, in bytes, of a task image file.
<WARNIN>	Value of the exit status code returned by a task that has issued warning diagnostic messages.
<CLI>	Always assigned the acronym MCR.
<DATE>	Assigned the current date; format is dd-mmm-yy.
<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).
<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.



Symbol	Value
<SYDISK>	Assigned the device mnemonic (two letters) of the user's default device (SY); format is dd (for example, DU).
<SYSUIC>	Assigned the system UIC; format is [ggg,mmm].
<TIME>	Assigned the current time; format is hh:mm:ss.
<UIC>	Assigned the current UIC. On IAS systems, <UIC> always contains your default UIC in the form [ggg,mmm].
<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, 3.4).

## 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<sub>8</sub> (0 to 65,535<sub>10</sub>), 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 displays:

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

A numeric symbol or constant can 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 does 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<sub>8</sub>, whereas the directive statements

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

assign numeric symbol N3 the value 16<sub>8</sub>.

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 .EXIT directive enables 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
```

## The Indirect Command Processor (Reference Section)

You can also assign a new value to a symbol with the `.ASKN` directive.

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.

### 2.4.3 String Symbols, Substrings, and Expressions

A string constant is a string of any printable characters enclosed by quotation marks ("). When you begin a string with a delimiter, you must end it with the same delimiter. You can also use empty strings. The number of characters cannot exceed 80<sub>10</sub>. For example:

```
"ABCDEF"
" "
```

String symbols can 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 can 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 DISPLAY Do you want to display the file?
```

displays the following question on the terminal:

## The Indirect Command Processor (Reference Section)

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

The symbol `DISPLY` is 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).

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 for an example of this behavior.)

With the `.GOSUB` directive, 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 MCR 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
```

## 2.5 Switches

Indirect accepts the switches listed in Table 2–1. You can use any combination of switches in the command line @filespec/switch(es) or in the .CHAIN filespec/switch(es).

These switches can be prefixed with a minus sign (–) or “NO” to negate the action of the switch (for example, /NOMCR suppresses sending commands to MCR for execution).

The switches specified in command line @filespec/switch(es) are used as defaults within the @filespec command or .CHAIN directive. This default-processing does not apply to the /DE switch; for that switch, the default is always /NODE.

**Table 2–1 Indirect Switches**

Switch	Default	Function
/TR	/NOTR	Trace—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 MCR commands issued as a result of the previous directive. The default is /NOTR.
/MCR	/MCR	MCR—Indicates that commands are to be passed to MCR.
/DE	/NODE	Delete—Indicates that the executing command file is to be deleted when processing is complete.
/LI	/LI	List—Indicates that MCR commands and Indirect comments are displayed on the terminal. This switch does not override the .ENABLE/.DISABLE LIST directive.

You can use any combination of the switches in the command line @filespec/switch(es) or in the directive statement .CHAIN filespec/switch(es).

## 2.6 Description of Indirect Directives

Directives must be separated from their arguments and from MCR-specific commands by at least one space. Unless you are using the .IF directives, only one directive is permissible 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 enables 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: The .DISABLE DATA statement must begin in the first column or Indirect places 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.**

---

## **.label:—Define a label**

Labels always appear at the beginning of the line. They can be on a line with additional directives and/or a MCR 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 can 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 enables 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, use the .ENABLE LRU directive and leave the line following the colon blank.

---

### **EXAMPLE(S)**

```
.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.

## .ASK

---

### .ASK—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.

---

### FORMAT

**.ASK** *ssssss txt-strng*

**.ASK** [*default:*] *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.

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

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.
- 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.

---

### EXAMPLE(S)

The directive statement:

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

displays the following text:

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



on the terminal. Symbol `PRINT` is 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).

---

## **.ASKN—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.

---

### **FORMAT**

**.ASKN** *ssssss txt-strng*

**.ASKN** [*low:high:default*] *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 enabling it to time out or by pressing the RETURN key.

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

**NOTE: If you omit any of the parameters within the square brackets, any preceding colons are required for positional identification.**

The command line cannot exceed 80<sub>10</sub> 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 is to be taken as octal or with [D]: to indicate that the response is to be taken as decimal. The reply must be a number either within the specified range or in the range 0 to 177777<sub>8</sub> (by default) or 0 to 65,535<sub>10</sub>.

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

AT. -- Value or string is out of range

Indirect then repeats the query.

If an arithmetic operation yields a result greater than 177777<sub>8</sub> 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 can 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

```
.ASKN A Enter value
```

---

## EXAMPLE(S)

The directive statement:

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

displays the following on the terminal:

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

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

Indirect then defines symbol SYM according to the reply entered.

In this next example, the directive statement:

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

displays the following on the terminal:

```
* Define numeric symbol A [O R:2-35 D:16]:
```

The format used in this display is as follows:

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

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.

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
```

## .ASKN

displays the following on the terminal (assuming the value of 16<sub>8</sub> for NUMSYM):

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

---

## **.ASKS—Ask for a string symbol definition.**

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 displays:

```
AT. -- Value or string is out of range
```

Indirect then repeats the query.

---

### **FORMAT**

**.ASKS** *ssssss txt-strng*

**.ASKS** [*low:high*] *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.

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.

---

### **EXAMPLE(S)**

The directive statement:

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

## .ASKS

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] MIDNAM Please enter your middle name
```

displays the following on the terminal:

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

The format used in this display is as follows:

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

where:

S            The symbol type (string).

R:low-high    The specified range for the number of characters.

---

## **.CHAIN—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 and .ONFAIL 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**

**.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(S)**

```
.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
```

## **.CLOSE**

---

### **.CLOSE—Close secondary file.**

The .CLOSE directive closes the secondary file opened by an .OPEN directive.

---

#### **FORMAT**

**.CLOSE** [*#n*]

---

#### **PARAMETERS**

##### ***n***

An optional file number in the range 0 to 3. The default is #0. You can substitute a numeric symbol for the value *n* by enclosing the symbol in apostrophes.



---

## .DATA—Output data to secondary file.

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

**.DATA** [#n] *txt-strings*

---

### PARAMETERS

***n***

An optional file number in the range 0 to 3. The default is #0. You can substitute a numeric symbol for the value *n* by enclosing the symbol in apostrophes.

***txt-strings***

The text to be output to the secondary file.

The command line cannot exceed 132<sub>10</sub> 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(S)

## **.DATA**

```
.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—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

---

### **PARAMETERS**

#### **SSSSSS**

The 1- to 6-character numeric symbol.

---

### **EXAMPLE(S)**

```
.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 causes an underflow error.

## .DELAY

---

# .DELAY—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<sub>10</sub> seconds

#10S = 10<sub>8</sub> 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(S)

```
.DELAY 20M
```

This directive statement delays processing for 20<sub>10</sub> minutes.

---

## **.DISABLE—Disable option.**

The **.DISABLE** directive disables a specified operating mode previously activated by a **.ENABLE** directive.

---

### **FORMAT**

**.DISABLE** *option[,option...]*

---

### **PARAMETERS**

#### ***option***

One or more of the operating modes described with the **.ENABLE** directive.

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

DATA	DECIMAL	DISPLAY	ESCAPE
GLOBAL	LIST	LOWERCASE	LRU
QUIET	STATISTICS	SUBSTITUTION	TIME

---

## .ENABLE—Enable option.

You can use the `.ENABLE` directive 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:

DATA	disabled (I)
DECIMAL	disabled (R)
DISPLAY	enabled (R)
ESCAPE	disabled (I)
GLOBAL	disabled (I)
LOWERCASE	enabled (I)
QUIET	disabled (R)
SUBSTITUTION	disabled (I)

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: `DATA`, `ESCAPE`, `GLOBAL`, `LOWERCASE`, and `SUBSTITUTION`. The remaining operating modes retain (denoted by “R” in the previous list) their new settings in the lower-level file.

In **DATA** mode, Indirect outputs lines that follow an `.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 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 **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 **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 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`, 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 `MCR` 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 (sssss of **.ASK** ssssss txt-string) is also set to true.

In **LIST** mode, commands display on the terminal. The list option can be used to override the **/LI** or **/-LI** switches.

In **LRU** mode, Indirect collects direct-access labels in a table for efficient access. This option requires that such labels be unique within a file. (See the **.label:** section for details about direct-access labels.)

In **QUIET** mode, Indirect does not echo MCR 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 **STATISTICS** mode, Indirect collects certain statistics and displays them when it exits.

In **TIME** mode, Indirect provides a time stamp with **MCR>** prompts.

## FORMAT

**.ENABLE** *option*

**.ENABLE DATA** [*#n*]

## PARAMETERS

### *option*

One of the operating modes described previously:

```
DATA
DECIMAL
DISPLAY
ESCAPE
GLOBAL
LIST
LOWERCASE
LRU
QUIET
STATISTICS
SUBSTITUTION
TIME
```

### *#n*

An optional file number in the range 0 to 3. The default is #0. You can substitute a numeric symbol for the value *n* by enclosing the symbol in apostrophes.

# .ENABLE

---

## EXAMPLE(S)

### ❶ 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 
$ PRINT SOURCES
```

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

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

### ❸ QUIET mode (MCR is the CLI for the terminal):

```
.ASK QUIET Do you want command lines suppressed
.IFT QUIET .ENABLE QUIET
.IFF QUIET .DISABLE QUIET
ACT /ALL
```

If the response is affirmative, Indirect displays the active tasks but not the ACT /ALL command. For example:

```
> * Do you want command lines suppressed? [Y/N]: (Y)
```



---

## **.EXIT—Exit from current command file.**

The .EXIT directive terminates processing of the current command file and returns control to the previous-level command file. If the directive is encountered at the uppermost indirect nesting level, Indirect exits and passes control to MCR.

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

---

### **FORMAT**

**.EXIT**

---

### **PARAMETERS**

#### ***value***

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

---

### **EXAMPLE(S)**

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>.

---

## **.GOSUB—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.

The maximum nesting depth for a subroutine call is eight.

---

### **FORMAT**

**.GOSUB** *label parameters*

---

### **PARAMETERS**

#### *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 to obtain formal call parameters.

---

### **EXAMPLE(S)**

```
.GOSUB EVAL
```

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

---

## **.GOTO—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.

See the `.label:` section for more information on labels and direct-access labels.

---

### **FORMAT**

**.GOTO** *label*

---

### **PARAMETERS**

#### ***label***

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

---

### **EXAMPLE(S)**

```
.GOTO 100
```

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

---

## **.IF—Test if symbol meets specified condition.**

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.

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.

---

### **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.

---

**EXAMPLE(S)**

```
.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.

## .IFACT/.IFNACT

---

### .IFACT/.IFNACT—Test if task is active or dormant.

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.

---

#### FORMAT

**.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.

---

#### EXAMPLE(S)

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

---

## .IFDEV/.IFNDEV—Test if device is present in the system.

The .IFDEV and .IFNDEV directives test whether a PUD exists (.IFDEV) or does not exist (.IFNDEV) for a device in the system. If the results of the test is true, the remainder of the command line is processed.

If an /H switch is appended to the <DEVICE> parameter, an additional check is made to see if the device physically exists by testing for the presence of the device's control status register (CSR).

---

### FORMAT

**.IFDEV** <DEVICE> *command-line*

**.IFNDEV** <DEVICE> *command-line*

**.IFDEV** <DEVICE>/H *command-line*  
**.IFNDEV** <DEVICE>/H *command-line*

---

### EXAMPLE(S)

A command file IFDEV.CMD contains the following commands:

```
.ENABLE SUBSTITUTION
.ENABLE LOWERCASE

.sets s1 "A PUD for"
.sets s2 "is defined in the system."
.sets s3 "The Controller for"
.sets s4 "is found in the system."
;
; Show the actual states of the MU devices.
; Note: Device MU2: has been generated into this system
;       (created a PUD) but the device does not physically exist.
;
DEV MU
;
.IFDEV MU0: ; 'S1' MU0: 'S2'
.IFDEV MU1: ; 'S1' MU1: 'S2'
.IFDEV MU2: ; 'S1' MU2: 'S2'
;
;
.IFDEV MU0:/H ; 'S3' MU0: 'S4'
.IFDEV MU1:/H ; 'S3' MU1: 'S4'
.IFDEV MU2:/H ; 'S3' MU2: 'S4'
/
```

When the file is executed, Indirect displays the following information:

## .IFDEV/.IFNDEV

```
MCR>@IFDEV
>;
>; Show the actual states of the MU devices.
>; Note: Device MU2: has been generated into this system
>;      (created a PUD) but the device does not physically exist.
>;
>DEV MU
  MU2
  MU1  **
  MU0  **
>;
>; A PUD for MU0: is defined in the system.
>; A PUD for MU1: is defined in the system.
>; A PUD for MU2: is defined in the system.
>;
>;
>; The Controller for MU0: is found in the system.
>; The Controller for MU1: is found in the system.
>/
@ <EOF>
```



---

## .IFDF/IFNDF—Test if symbol is defined or not defined.

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
.
.
.
```

---

### FORMAT

**.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.

---

### EXAMPLE(S)

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

---

## .IFFILE/.IFNFILE—Test for the existence of a file.

The .IFFILE and .IFNFILE directives determine whether a file exists (.IFFILE) or does not exist (.IFNFILE). If the result of the test is true, the remainder of the command line is processed.

---

### FORMAT

**.IFFILE** *<file specification> command-line*

**.IFNFILE** *<file specification> command-line*

Restriction:

Both directives fail when the directory for a specified file does not exist and does not generate an "AT.—Syntax error" message. Perform a separate test for the directory before you use this directive when you are not certain that the directory exists.

---

### EXAMPLE(S)

A command file IFFILE.COMD contains the following commands:

```
.ENABLE SUBSTITUTION
.ENABLE LOWERCASE

.SETS FILE1 "[1,1]STARTUP.COMD"
.SETS FILE2 "[377,300]HOT.TUB"
;
;
;
; Show .IFFILE behavior when files do and do not exist.
;
.IFFILE 'FILE1'           ; 'FILE1' is found
.IFFILE 'FILE2'           ; 'FILE2' is found
; Show .IFNFILE behavior when files do and do not exist
; (but directory does exist).
.IFNFILE 'FILE1'          ; 'FILE1' is not found
.IFNFILE 'FILE3'          ; 'FILE3' is not found
; Show how to use .IFNFILE when a directory might not exist.

.; Call subroutine to build a MFD specification.
.GOSUB MUNG
.IFNFILE 'FINAL' .GOTO 10:
.IFNFILE 'FILE2'          ; 'FILE2' is not found
.GOTO EXIT
.10:                       ; Directory for 'FILE2' does not exist
.EXIT:
/
```

```
.;-----  
.;  
.; Subroutine MUNG  
.; Create a MFD file specification from a file specification.  
.;  
.; Input: FILE2 containing file specification, including UFD.  
.; Output: FINAL contains MFD specification for UFD in FILE2  
.;  
.;-----  
.MUNG:  
  
.; Convert the UFD from the file specification into a MFD file name.  
.; * Set up the constant portions of the final string  
.; * Locate the comma in the UFD (eg: [xxx,yyy])  
.; * Extract the group and member code  
.; * Build the final MFD file syntax  
  
.SETS MFD "[0,0]"  
.SETS EXT ".DIR"  
  
.SEARCH "'FILE2'" "," COMMA  
  
.SETS DIR1 FILE2[2:'COMMA'-1]  
.SETS DIR2 FILE2['COMMA'+1:10]  
  
.SETS FINAL MFD+DIR1+DIR2+EXT  
  
.RETURN
```

When the file is executed Indirect displays the following information:

```
MCR>@X  
>;  
>;  
>;  
>; Show .IFFILE behavior when files do and do not exist.  
>;  
>; [1,1]STARTUP.CMD is found  
>;  
>; Show .IFNFILE behavior when files do and do not exist  
>;  
>; [1,1]WATER.BED is not found  
>;  
>; Show how to use directive when a directory might not exist.  
>;  
>; Directory for [377,300]HOT.TUB does not exist  
& <EOF>
```

And if the directory did exist the last >; line would read

```
>; [377,300]HOT.TUB is not found
```

---

## **.IFINS/.IFNINS—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.

---

### **FORMAT**

**.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.

---

### **EXAMPLE(S)**

```
.IFINS PIP .GOTO 250
.IFNINS PIP INS[11,1]PIP
```

---

## **.IFLOA/.IFNLOA—Test if handler is loaded or not loaded.**

The .IFLOA and .IFNLOA directives test whether a handler 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.

---

### **FORMAT**

**.IFLOA** <device>*directive-statement*

**.IFNLOA** <device>*directive-statement*

---

### **PARAMETERS**

#### **<device>**

A device handler

#### ***directive-statement***

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

---

### **EXAMPLE(S)**

```
.IFLOA DU: .GOTO 250
```

```
.IFNLOA DU: LOA DU:
```

---

## .IFMOU/.IFNMOU—Test if device is mounted or not mounted.

The .IFMOU and .IFNMOU directives test whether a device is mounted (.IFMOU) or not mounted (.IFNMOU) either FILES-11 or foreign. If the result of the test is true, the remainder of the command line is processed.

---

### FORMAT

**.IFMOU** <device>command line

**.IFNMOU** <device>command-line

---

### EXAMPLE(S)

A command file IFMOU.CMD contains the following commands:

```
; Show the actual states of the MU devices
;
  DEV MU
;
.IFMOU MU0:      ; Device MU0: is mounted.
.IFMOU MU1:      ; Device MU1: is mounted
;
;
.IFNMOU MU0:     ; Device MU0: is not mounted
.IFNMOU MU1:     ; Device MU1: is not mounted
/
```

When the file is executed Indirect displays the following information:

```
MCR>@IFMOU
>; Show the actual states of the MU devices
>;
>DEV MU
  MU1  **    Mounted    Global
  MU0  **
>;
>; Device MU1: is mounted
>;
>;
>; Device MU0: is not mounted
@ <EOF>
```

---

## .IFPAR/.IFNPAR—Test for memory partition.

The .IFPAR and .IFNPAR directives test whether a memory partition exists (.IFPAR) or does not exist (.IFNPAR). If the result of the test is true, the remainder of the command line is processed.

---

### FORMAT

**.IFPAR** <partition\_name> *command\_line*

**.IFNPAR** <partition\_name> *command\_line*

---

### EXAMPLE(S)

A command file IFPAR contains the following commands:

```
.IFPAR GEN           ; Partition GEN is found in the system
.IFPAR REAL          ; Partition REAL is found in the system

.IFNPAR GEN          ; Partition GEN is *not* found in the system
.IFNPAR REAL         ; Partition REAL is *not* found in the system
```

When the file is executed, Indirect displays the following information:

```
MCR>@IFPAR
>; Partition GEN is found in the system
>; Partition REAL is *not* found in the system
@ <EOF>
```

---

## **.IFT/.IFF—Test if symbol is true or false.**

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.

---

### **FORMAT**

**.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.

---

### **EXAMPLE(S)**

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



---

## .IFREADY/.IFNREADY—Test if specified disk is on or off.

The .IFREADY and .IFNREADY directives determine if a disk is online (.IFREADY) or offline (.IFNREADY). .IFREADY executes <command> if the tested condition is true. (These directives apply only to disk devices.)

Synonym directives are .IFNOFF (for .IFREADY) and .IFOFF (for .IFNREADY).

---

### FORMAT

**.IFREADY** <device:> <command>

**.IFNOFF** <device:> <command>

**.IFNREADY** <device:> <command>

**.IFOFF** <device:> <command>

---

### EXAMPLE(S)

A command file IFREADY.CMD contains the following commands:

```
.ENABLE SUBSTITUTION
.ENABLE LOWERCASE

.IFREADY DU0:           ; DU0: is on-line
.IFREADY DU4:           ; DU4: is on-line

.IFNREADY DU0:         ; DU0: is *not* on-line
.IFNREADY DU4:         ; DU4: is *not* on-line

;
; This shows the IFOFF/IFNOFF synonyms
;

.IFOFF DU0:            ; DU0: is *not* on-line
.IFOFF DU4:            ; DU4: is *not* on-line

.IFNOFF DU0:           ; DU0: is on-line
.IFNOFF DU4:           ; DU4: is on-line
```

When the file is executed Indirect displays the following information:

## .IFREADY/.IFNREADY

```
MCR>@IFREADY
>; DU0: is on-line
>; DU4: is *not* on-line
>;
>; This shows the IFOFF/IFNOFF synonyms
>;
>; DU4: is *not* on-line
>; DU0: is on-line
@ <EOF>
```

### 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 compound operators .AND and .OR must be preceded and followed by at least one blank space.

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
```

---

## EXAMPLE(S)

```
.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 PIP
```

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

```
.IF X EQ 3 .OR .IF Y LE Z .GOTO NEXT
```

Execute the .GOTO directive when X equals 3 or when Y is less than Z. If neither condition exists, do not execute the directive.

---

## **.INC—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

---

### **PARAMETERS**

#### **SSSSSS**

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

---

### **EXAMPLE(S)**

```
.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 causes an overflow error.

---

## **/—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(S)**

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

---

## **.ONERR—Detect directive errors and branch to a label.**

The .ONERR directive detects errors such as syntax errors in the Indirect command file and branches to a specified label.

Error trapping remains in effect until .ONERR is disabled. To disable .ONERR, specify the .ONERR directive without a label.

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, .SETN, .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 PIP syntax error) are not processed by the .ONERR directive.

---

### **FORMAT**

**.ONERR** [*label*]

---

### **PARAMETERS**

#### ***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.

## .ONERR

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

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

---

### EXAMPLE(S)

The ONERR.CMD command file contains the following commands:

```
.ENABLE SUBSTITUTION
.ENABLE LOWERCASE
.;
.; Define a lable for an error branch
.;
.ONERR SERR

;
; This .SETS directive generates a syntax error.
.; The correct syntax is .SETS TEXT "This is a test".
;
;
.SETS TEXT THIS IS A TEST
; This should never be executed.

.10:
.ONERR EERR
;
; This .ENABLE generates an error.
;
.ENABLE MAGIC
; This should never be executed.

.20:
.; Disable ONERR error traps
.ONERR

;
; This directive generates an error and Indirect terminates
; because ONERR error trapping is disabled.
;
.SETS TEST THIS IS ANOTHER TEST
;
; This line is NOT executed because error trapping is disabled.
;

.GOTO EXIT

.SERR:
;
; A .SETS error is detected.
;
.GOTO 10
```

```
.EERR:
;
; The .ENABLE directive specified an undefined function.
;
.GOTO 20
.EXIT:
```

When the file is executed, Indirect displays the following information:

```
MCR>@ONERR
>;
>; This .SETS directive generates a syntax error.
>;
>;
AT. -- UNDEF SYM=THIS
.SETS TEXT THIS IS A TEST
>;
>; A .SETS error is detected.
>;
>;
>; This .ENABLE generates an error.
>;
AT. -- SYNTAX ERR
.ENABLE MAGIC
>;
>; The .ENABLE directive specified an undefined function.
>;
>;
>; This directive generates an error and Indirect terminates
>; because ONERR error-trapping is disabled.
>;
AT. -- UNDEF SYM=THIS
.SETS TEST THIS IS ANOTHER TEST
```

**NOTE:** Indirect did not signal the end of the command file with @<EOF>.

---

## **.ONFAIL—Detect any MCR command that returns an error status.**

The .ONFAIL directive detects any MCR command that returns an error status (such as a MACRO command that has assembly errors) and branches to a specified label.

Error-trapping remains in effect until disabled. To disable .ONFAIL error detection specify the .ONFAIL directive without a label.

The .ONFAIL directive function is similar to the .ONERR directive, which detects Indirect Command File directive errors.

---

### **FORMAT**

**.ONFAIL<label>**

**.ONFAIL**

---

### **EXAMPLE(S)**

A command file ONFAIL.COMD contains the following commands:

```
.ENABLE LOWERCASE
-;
-; Trap assembly errors to the MACERR routine.
-;
.ONFAIL MACERR
    MAC TEST,TEST/--SP=TEST

.10:
-;
-; Trap task build errors to the TKBERR routine.
-;
.ONFAIL TKBERR
    TKB TEST,TEST/--SP=TEST

.20:
-;
-; The RUN command generates an error.
-;
    RUN TEST

;
;
;
; This line shows that error trapping is disabled because the above
; RUN command generated an error and the command file did not branch.
;

.GOTO EXIT

.MACERR:
```



```
;
; Macro assembly error detected. Resetting the error trap branch.
;
.; Perform the Task build to show how to change the error trap branch
.GOTO 10

.TKBERR:

;
; Task Builder error detected. Disable error trapping.
;

.ONFAIL
.GOTO 20

.EXIT:
```

When the file is executed Indirect displays the following information:

```
>MAC TEST,TEST/-SP=TEST
MAC -- Open failure on input file
TEST,TEST/-SP=TEST
>;
>; Macro assembly error detected. Resetting the error trap branch.
>;
>TKB TEST,TEST/-SP=TEST
TKB -- *FATAL*-File TEST.OBJ;23 has illegal format

>;
>; Task Builder error detected. Disable error trapping.
>;
>RUN TEST
>;
>;
>;
>;
INS -- OPEN FAILURE FILE TEST.TSK
>; This line shows that error trapping is disabled because the above
>; RUN command generated an error and the command file did not branch.
>;
@ <EOF>
```

## .OPEN

---

### .OPEN—Open secondary file.

The .OPEN directive opens a specified secondary file as an output file. The .DATA directive is used to place data in the secondary file opened by the .OPEN directive.

---

#### FORMAT

**.OPEN** [*#n*] *filespec*

---

#### PARAMETERS

##### **#n**

An optional file number in the range 0 to 3. The default is #0. To substitute a numeric symbol for the value n, enclose 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.

**NOTE: You cannot include comments in an .OPEN statement. Doing so results in a syntax error.**

---

#### EXAMPLE(S)

```
.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.

---

## **.OPENA—Open secondary file for append.**

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

---

### **FORMAT**

**.OPENA** [#*n*] *filespec*

---

### **PARAMETERS**

*n*

An optional file number in the range 0 to 3. 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.

**NOTE: You cannot include comments in an .OPENA statement. Doing so results in a syntax error.**

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

---

### **EXAMPLE(S)**

```
.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.

---

## .OPENR—Open file for reading.

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

---

### FORMAT

**.OPENR** [*#n*] *filespec*

---

### PARAMETERS

#### *n*

An optional file number in the range 0 to 3. 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.

**NOTE: You cannot include comments in an .OPENR statement. Doing so results in a syntax error.**

---

### EXAMPLE(S)

```
.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.

---

## .PARSE—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 a null substring comes 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).

---

### EXAMPLE(S)

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

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

When the file is executed Indirect displays the following information:

```
MCR>@PARSE THIS IS A TEST OF THE EMERGENCY BROADCASTING SYSTEM.
>;
>; COMMAN    = PARSE
>; A1       = THIS
>; A2       = IS
>; A3       = A
>; A4       = TEST
>; A5       = OF THE EMERGENCY BROADCASTING SYSTEM.
>; <STRLEN> = 6
@ <EOF>
```

## **.PAUSE**

---

### **.PAUSE—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.

---

#### **FORMAT**

##### **.PAUSE**

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

```
AT. -- Pausing, 2 cntnue type "RES ...AT.'"<ALTMODE>
```

##### ***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
```

---

**.READ—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 117<sub>10</sub> characters, an error occurs.

**.READ** does not detect End of File.

---

**FORMAT**

**.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 is to be read.

---

**EXAMPLE(S)**

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**

---

### **.RETURN—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**



---

## **.SEARCH—Search a string for location of substring.**

The .SEARCH directive determines the position of a substring within a given string.

---

### **FORMAT**

**.SEARCH** <string> <substring> <variable>

The directive searches the string for the specified substring. A character count specifying the starting location of the substring is returned in the numeric <variable>.

If <variable> is not yet defined, it is defined according to the current radix. A zero (0) is returned in <variable> if the substring is not found.

The SEARCH directive is case-sensitive.

---

### **EXAMPLE(S)**

A command file SEARCH.CMD contains the following command lines:

```
.ENABLE SUBSTITUTION
.ENABLE LOWERCASE
.ENABLE DECIMAL

.SETS STRING "This is a test of the Emergency Broadcasting System"
.SETS SUBSTR "Emergency"
;
; The full text string is:
;   'STRING'
;
; The search string is:
;   'SUBSTR'
;
.; Now performing the search... START is a previously undefined variable
.SEARCH STRING SUBSTR START
; The word 'SUBSTR' starts at character # 'START'.
;
```

The result of executing the file is:

```
MCR>@SEARCH
>;
>; The full text string is:
>;   This is a test of the Emergency Broadcasting System
>;
>; The search s>;   Emergency
>;
>; The word Emergency starts at character # 23.
>;
@ <EOF>
```

---

## **.SETT/.SETF—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.

---

### **FORMAT**

```
.SETT  SSSSSS  
.SETF  SSSSSS
```

---

### **PARAMETERS**

#### **SSSSSS**

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

---

### **EXAMPLE(S)**

```
.SETT X
```

This directive sets the logical symbol X to true.

```
.SETF ABCDE
```

This directive sets the logical symbol ABCDE to false.

---

## **.SETN—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 can 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.

---

### **EXAMPLE(S)**

```
.SETN NUMBER 27
```

This directive assigns to the numeric symbol NUMBER the value 27<sub>8</sub>.

```
.SETN A2 10.  
.SETN A1 3*(A2-5)
```

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

---

## .SETS—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** *sssss strexp*

---

### PARAMETERS

#### **sssss**

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”).

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.

---

### EXAMPLE(S)

```
.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 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).

A command file SETS1.CMD contains the following commands:

```
.ENABLE SUBSTITUTION
.SETS A      "ABCDEF"          ; ASSIGN QUOTED STRING TO VARIABLE A
.SETS STR2  "ZZZZ"           ; ASSIGN QUOTED STRING TO VARIABLE STR2
.SETS X      STR2+"ABC"       ; APPEND QUOTED STRING TO VARIABLE STR2
.SETS Z      STR2+A[4:6]     ; APPEND LAST THREE CHRACTERS OF VARIABLE A
.;                               ; TO THE END OF VARIABLE STR2
.SETS MYFILE <UIC>+"MYFILE.TXT" ; PREFIX THE QUOTED STRING WITH THE CURRENT
.;                               ; USER IDENTIFICATION CODE

;
; HERE ARE THE RESULTS
;
; VARIABLE      VARIABLE
; NAME          CONTENTS
;-----
;
;      A          'A'
;   STR2        'STR2'
;      X          'X'
;      Z          'Z'
; MYFILE        'MYFILE'
;
```

When the file is executed, Indirect displays the following information:

```
MCR>@SETS
>;
>; HERE ARE THE RESULTS
>;
>; VARIABLE      VARIABLE
>; NAME          CONTENTS
>;-----
>;
>;      A          ABCDEF
>;   STR2        ZZZZ
>;      X          ZZZZABC
>;      Z          ZZZZDEF
>; MYFILE        [1,1]MYFILE.TXT
>;
@ <EOF>
```

A command file SETS2.CMD contains the following commands:

```
.ENABLE SUBSTITUTION
.; Convert the UFD from the file specification into a MFD file name.
.; First, set up the constant portions of the MFD string
.; Second, locate the comma in the UFD (eg: [xxx,yyy])
.; Third, extract the group and member code
.; Fourth build the complete MFD file syntax

.SETS FILESPEC "[377,300]HOT.TUB"
.SETS MFD "[0,0]"
.SETS EXT ".DIR"

.SEARCH "'FILESPEC'" "," COMMA

.SETS DIR1 FILESPEC[2:'COMMA'-1]
.SETS DIR2 FILESPEC['COMMA'+1:10]

.SETS MFDSPEC MFD+DIR1+DIR2+EXT
```

## .SETS

```
;
; The MFD file specification for ['dir1','dir2'] is 'MFDSPEC'
;
```

When the file is executed, Indirect displays the following information:

```
>;
>; The MFD file specification for [377,300] is [0,0]377300.DIR
>;
@ <EOF>
```

---

## **.TASK—Read task file and store size.**

The **.TASK** directive determines the size of a task image file and stores the size in the numeric variable **<TSKTSZ>**.

Indirect exits if the specified file does not exist. However, you can use the **.ONERR** directive to continue processing.

---

### **FORMAT**

**.TASK** *<Task image file specification>*

---

### **EXAMPLE(S)**

A command file **TASK.COMD** contains the following commands:

```
.ENABLE SUBSTITUTION
.ENABLE LOWERCASE

.; Trap "File not found errors" else IND will exit.
.ONERR RETRY

.PROMPT:
.ASKS FILE Specify a task image
.TASK 'FILE'

;
; The size of 'FILE' is '<TSKTSZ>' decimal bytes.
;
/

.RETRY:
;
; Unable to open 'FILE'. Please enter a different file specification
;
.GOTO PROMPT:
```

When the file is executed Indirect displays the following information:

```
MCR>@TASK
>* Specify a task image [S]: [11,1]TKB.TSK
>;
>; The size of [11,1]TKB.TSK is 16832 decimal bytes.
>;
>/
@ <EOF>
```

## .TEST

---

### .TEST—Test symbol.

The .TEST directive determines the length of a string symbol and stores the length of the string in the special numeric symbol <STRLEN>. It also tests the characters of the string and sets the special logical symbols <ALPHAN>, <RAD50>, and <OCTAL> accordingly.

---

#### FORMAT

**.TEST** <*string\_symbol*>

---

#### PARAMETERS

##### <STRING\_SYMBOL>

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

The results of the test are as follows:

- If variable is a string, <STRLEN> contains the length of the string. Also, the special symbols <ALPHAN> and <RAD50> are set based on a scan of the characters of variable.
- If variable is an octal value, <OCTAL> is set to TRUE.

---

#### FORMAT

**.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”).

---

#### EXAMPLE(S)

If SUM is a string symbol, the directive statement

```
.TEST SUM
```

places the number of characters represented by the symbol SUM into <STRLEN>.



A command file TEST.CMD contains the following commands:

```

.ENABLE SUBSTITUTION

.SETS A "THIS IS A TEST"
.TEST A
.GOSUB SHOW
;
; Note: The string 'A' contains a blank space which is
;       neither an alphanumeric nor RAD50 value.
;

.SETS A "$.$.$.$."
.TEST A
.GOSUB SHOW

.SETS A "12345"
.TEST A
.GOSUB SHOW

.SETS A "!@#%$^&*()_+"
.TEST A
.GOSUB SHOW

/

.SHOW:
;
; Test string = 'A'
;
;
; Special symbol           Special symbol value
;-----
; STRLEN   =               '<STRLEN>'
; ALPHAN   =               '<ALPHAN>'
; RAD50    =               '<RAD50>'

.RETURN

```

When the file is executed Indirect displays the following information:

```

MCR>@test2
>;
>; Test string = THIS IS A TEST
>;
>;
>; Special symbol           Special symbol value
>;-----
>; STRLEN   =               16
>; ALPHAN   =               FALSE
>; RAD50    =               FALSE
>;
>; Note: The string THIS IS A TEST contains a blank space which is
>;       neither an alphanumeric nor RAD50 value.
>;
>;
>; Test string = $.$.$.$
>;
>;
>; Special symbol           Special symbol value
>;-----
>; STRLEN   =               10
>; ALPHAN   =               FALSE
>; RAD50    =               TRUE

```

# .TEST

```
>;
>; Test string = 12345
>;
>;
>; Special symbol          Special symbol value
>;-----
>; STRLEN   =              5
>; ALPHAN   =              TRUE
>; RAD50    =              TRUE
>;
>; Test string = !@#$$%^&*()_+
>;
>;
>; Special symbol          Special symbol value
>;-----
>; STRLEN   =              14
>; ALPHAN   =              FALSE
>; RAD50    =              FALSE
>/
>@ <EOF>
```

---

## **.WAIT—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*

---

### **PARAMETERS**

#### ***taskname***

A 1- to 6-character valid task name.

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 /NOMC switch is in effect.

---

### **EXAMPLE(S)**

```
.WAIT xxx
```

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

---

## **.XQT—Initiate parallel task execution.**

Indirect usually passes a command to MCR 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 enables you to start a task, to pass a command line to it, and to continue processing in parallel with the initiated task. The maximum number of successive .XQT directives allowed is 10 (decimal).

---

### **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 enables you to initiate parallel processing of tasks. The .WAIT directive is used to synchronize their execution.

A 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 Using an Indirect Command File

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

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

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

```
> @PRINTER 
```

### 2.7.2 Asking for a Device Specification

```
.;
.; This command file asks for a device specification.
.; You can 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 ":"
.TEST DEVICE
.IF TEMPN EQ <STRLEN> .SETS DEVICE DEVICE+"0"
.IF TEMPS NE DEVICE[<STRLEN>:<STRLEN>] .SETS DEVICE DEVICE+":"
; The full device specification is 'DEVICE'
.EXIT
```

①  
②  
③  
④  
⑤  
⑥  
⑦  
⑧  
⑨  
⑩

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

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

- ① Substitution mode enabled.
- ② Lowercase mode disabled, which means that all input characters are converted to uppercase regardless of how they were typed in.
- ③ Asks for the device name (that is, the mnemonic and unit number) and assigns it to the string symbol DEVICE.
- ④ Sets numeric symbol TEMPN to the value 2, which is the number of characters for the device mnemonic.

## The Indirect Command Processor (Reference Section)

- ⑤ Sets string symbol `TEMPS` to contain a colon. The colon is a string constant, so it must be enclosed in quotation marks.
- ⑥ Tests the symbol `DEVICE` (which contains the specified device name). As a result, the following special symbols are set:  
`<STRLEN>` = The length of the string (the number of characters typed in response to the question)
- ⑦ 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`.
- ⑧ 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:`).
- ⑨ Displays this text, with the full device name substituted for '`DEVICE`,' on the terminal.

### 2.7.3 Initializing and Mounting a Volume, and Copying Files to That Volume

```
.ENABLE SUBSTITUTION
.GETDEV:
.ASKS DEVICE Enter device (DU1 or DU2)
.IF DEVICE EQ "DU0" .GOTO GETDEV
.ASKS DIR What directory (include square brackets)?
.INIT:
.ASK INIT Initialize device
.IFF INIT .GOTO COPY
ALLOCATE 'DEVICE':
MOUNT/FOREIGN 'DEVICE'
.ASKS LABEL What volume label?
INITIALIZE 'DEVICE': 'LABEL'
DISMOUNT/NOUNLOAD 'DEVICE':
MOUNT/NOSHAREABLE 'DEVICE': 'LABEL'
CREATE/DIRECTORY 'DEVICE': 'DIR'
.COPY:
.ASKS FILES Enter names of files (file1,file2,...)
COPY 'FILES' 'DEVICE': 'DIR'
.ASK MORE More files
.IFT MORE .GOTO COPY
.ASK LIST List directory
.IFF LIST .GOTO END
DIRECTORY 'DEVICE': 'DIR'
.END:
DISMOUNT 'DEVICE':
DEALLOCATE 'DEVICE':
.EXIT
```

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

- ① Substitution mode enabled.
- ② Line for `.GETDEV:` label. It is a direct-access label, so it is the only element on the command line.
- ③ Asks for the name of the device to which the files are to be copied.
- ④ Performs a conditional test. If `DEVICE = DU0` (an invalid device), returns to `.GETDEV:` and asks the question again.
- ⑤ Asks for the directory to which the files are to be copied.

- ⑥ Line for the .INIT: label (also a direct-access label).
- ⑦ Asks if the device should be initialized.
- ⑧ If the device should not be initialized, proceeds with the copy operation.
- ⑨ Allocates the specified device.
- ⑩ Mounts the device foreign, which is necessary for initializing a device.
- ⑪ Asks for the label for the volume.
- ⑫ Initializes the volume and gives it the specified label.
- ⑬ Dismounts the device without spinning it down.
- ⑭ Remounts the device as a private, Files-11 volume.
- ⑮ Creates the specified directory on the volume.
- ⑯ Line for the .COPY: label (also a direct-access label).
- ⑰ Asks for the specifications of the files to be copied.
- ⑱ Copies the files to the device.
- ⑲ Asks if there are more files to be copied.
- ⑳ 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.

### 2.7.4 Editing, Purging, Printing, and Formatting Files

.ENABLE QUIET	①
.ENABLE SUBSTITUTION	②
.ASKS FILNAM What is the file name?	③
.ASKS FILTYP What is the file type?	④
EDIT 'FILNAM'.'FILTYP'	⑤
.ASK A Do you want to purge this file?	⑥
.IFT A PIP 'FILNAM'.'FILTYP'/QU:2	⑦
PIP 'FILNAM'.'FILTYP';*/TR	⑧
.ASK DSR Do you want to invoke DSR?	⑨
.IFT DSR .GOSUB PROC	⑩
.ASK B Do you want a listing?	⑪
.IFF B .GOTO 100	⑫
.GOSUB LIST	⑬
QUE 'FILNAM'.'FILTYP'/CO:'C'/CO:'D'	⑭

## The Indirect Command Processor (Reference Section)

```
.100:                                     15  
      .EXIT                               16  
.PROC:                                     17  
      DSR 'FILNAM'='FILNAM'              18  
      .SETS FILTYP "MEM"                 19  
      .ASK F Do you want to purge the .MEM files? 20  
      .IFT F PIP 'FILNAM'.MEM/PU:2       21  
      PIP 'FILNAM'.MEM;*/TR              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) MCR 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, PIP 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.
- ⓵ If you do want to purge the files, PIP does so, keeping the two latest versions of the file.
- ⓶ Truncates the files to free up blocks that are allocated to the files but not used.
- ⓷ Returns to the line after .GOSUB PROC (.ASK B Do you want a listing).
- ⓸ Line for .LIST: label (label for the listing subroutine).



## The Indirect Command Processor (Reference Section)

- ②⑤ Asks for the form number for the line printer. Sets the numeric symbol C to this value, which is used in the QUE 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 QUE command line.
- ②⑦ Returns to the line after .GOSUB LIST (the QUE command line).



---

# 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. 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:** An illegal character was specified as a range or default argument. Only numeric expressions are permitted.

## Indirect Messages

AT — Command file open error

**Explanation:** 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:** 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:** 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:** A .OPEN or .OPENA directive specified a file that was already open.

AT. — File attributes not available

**Explanation:** An attempt was made to obtain file-attribute information with the <FILATR> symbol before any files were opened.

AT. — File not found

**Explanation:** An @filename or .CHAIN directive specified an incorrect file name or nonexistent file.

AT. — File not open

**Explanation:** Indirect encountered a .DATA or .CLOSE directive that did not reference an open file.

AT. — File read error

**Explanation:** An error was detected in reading the indirect command file. This error is usually caused by records that are more than 132<sub>10</sub> bytes long.

AT. — Illegal file number

**Explanation:** 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:** 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. — Invalid keyword

**Explanation:** An unrecognized keyword (preceded by a period) was specified.

AT. — Label not at beginning of line

**Explanation:** The specified label does not start in the first column of the line. All labels must do so.

AT. — Maximum indirect file depth exceeded

**Explanation:** 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. — Numeric under- or overflow

**Explanation:** The evaluation of a numeric expression yielded a value outside the range 0 to 177777<sub>8</sub>. This means that the value crossed the zero boundary from positive to negative or negative to positive.

AT. — Redefining symbol to different type <SSSSSS>

**Explanation:** 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:** A .RETURN directive was specified without a previous call to a subroutine (.GOSUB).

AT. — Spawn failure

**Explanation:** Indirect could not initiate the execution of a user command task.

AT. — String expression larger than 132. bytes

**Explanation:** An attempt was made to generate a string expression longer than 132<sub>10</sub> characters.

AT. — String substitution error

**Explanation:** 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:** 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:** The symbol table was full and there was no space for symbol ssssss.

AT. — Symbol type error <SSSSSS>

**Explanation:** 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:** The format of the specified command line is incorrect.

AT. — Too many concurrent .XQTs

**Explanation:** More than the maximum number of successive .XQT directives allowed by the build file for the Indirect task were issued.

## Indirect Messages

AT. — Undefined label <.label:>

**Explanation:** The label .label: specified with a .GOTO, .GOSUB, or .ONERR directive could not be found.

AT. — Undefined symbol <SSSSSS>

**Explanation:** The symbol ssssss was referenced, but it had not been defined.

---

# Index

---

---

## A

---

<ALPHAN> symbol • 2-7  
<ALTMOD> symbol • 2-7  
Arithmetic operator • 2-9  
.ASK directive • 2-16  
.ASKN directive • 2-18  
.ASKS directive • 2-21  
...AT. • 2-2  
At sign (@) • 1-1, 2-1, 2-2

---

## B

---

Begin-End block processing  
terminating • 2-33

---

## C

---

.CHAIN directive • 2-23  
<CLI> symbol • 2-8  
.CLOSE directive • 2-24  
Command line  
parsing • 2-61  
COMMAN symbol • 2-12  
Comment • 1-5, 2-3  
Compound test • 2-50

---

## D

---

.DATA directive • 2-25  
Data mode • 2-30  
<DATE> symbol • 2-8  
.DEC directive • 2-27  
Decimal mode • 2-30  
<DEFAUL> symbol • 2-7  
.DELAY directive • 2-28  
/DE switch • 2-13  
Device handler  
testing • 2-45, 2-46  
Direct-access label • 2-15

Directive • 1-2, 2-2  
functions • 2-2  
summary • 2-4 to 2-6

Directives  
description of • 2-14  
.DISABLE directive • 2-29  
Display mode • 2-30

---

## E

---

.ENABLE directive • 2-30  
.ENABLE GLOBAL directive • 2-3  
<EOF> symbol • 2-7  
<ERROR> symbol • 2-8  
Error messages • A-1 to A-4  
Error processing • 2-53  
<ESCAPE> symbol • 2-7  
Escape mode • 2-31  
Examples • 1-5, 2-77 to 2-81  
.EXIT directive • 2-33  
Exit status • 2-8  
value • 2-8  
<EXSTAT> symbol • 2-8

---

## F

---

<FALSE> symbol • 2-7  
File  
opening for reading • 2-60

---

## G

---

Global mode • 2-30  
.GOSUB directive • 2-34  
.GOTO directive • 2-35

---

## I

---

<IAS> symbol • 2-7  
.IFACT directive • 2-38

# Index

.IFDEV directive • 2–39  
.IFDF directive • 2–41  
.IF directive • 2–36  
.IFF directive • 2–48  
.IFFILE directive • 2–42  
.IFINS directive • 2–44  
.IFLOA directive • 2–45  
.IFMOU directive • 2–46  
.IFNACT directive • 2–38  
.IFNDEV directive • 2–39  
.IFNDF directive • 2–41  
.IFNFILE directive • 2–42  
.IFNINS directive • 2–44  
.IFNLOA directive • 2–45  
.IFNMOU directive • 2–46  
.IFNPAR directive • 2–47  
.IFNREADY directive • 2–49  
.IFPAR directive • 2–47  
.IFREADY directive • 2–49  
.IFT directive • 2–48  
.INC directive • 2–51  
Indirect • 1–1, 2–1  
Indirect command file • 2–1  
    chaining • 2–23  
    CLI  
        nesting • 2–2  
    formatting • 2–14  
    MCR • 2–2  
        default file type • 2–2  
    task • 2–1  
        default file type • 2–1  
        nesting • 2–1  
    tracing • 2–13  
    using task name • 2–4  
Indirect command file processing  
    delaying • 2–28  
    interrupting • 2–62  
    suspending • 2–75  
    terminating • 2–33, 2–52  
Indirect Command Processor  
    See Indirect

---

## L

Label • 1–3  
    branching to • 2–35  
    defining • 2–15  
    direct-access • 2–15  
<LIBUIC> symbol • 2–8

List mode • 2–31  
/LI switch • 2–13  
Logical end-of-file directive • 2–52  
Logical operator • 2–9, 2–11  
Logical symbol • 2–6  
    defining • 2–16  
    setting • 2–66  
    testing • 2–41, 2–42, 2–47, 2–48, 2–49  
Logical test • 2–36  
/LO switch • 2–3  
Lowercase mode • 2–30  
LRU mode • 2–31

---

## M

<MAPPED> symbol • 2–7  
/MCR switch • 2–3, 2–13  
<MEMSIZ> symbol • 2–8

---

## N

<NETUIC> symbol • 2–8  
Numeric expression • 2–9  
Numeric symbol • 2–7, 2–9  
    comparing • 2–36  
    decrementing • 2–27  
    defining • 2–18  
    incrementing • 2–51  
    radix • 2–9  
    setting • 2–67  
    substituting • 2–10  
    testing • 2–41, 2–42

---

## O

<OCTAL> symbol • 2–7  
.ONERR directive • 2–53  
.OPENA directive • 2–59  
.OPEN directive • 2–58  
.OPENR directive • 2–60  
Operating mode  
    defaults • 2–30  
    disabling • 2–29  
    enabling • 2–30  
    list • 2–30



---

## P

---

.PARSE directive • 2-61  
 .PAUSE directive • 2-62

---

## Q

---

Quiet mode • 2-31

---

## R

---

<RAD50> symbol • 2-8  
 .READ directive • 2-63  
 Record  
   reading • 2-63  
 Reserved symbol • 2-12  
 .RETURN directive • 2-64  
 <RSX11D> symbol • 2-8

---

## S

---

Secondary file  
   closing • 2-24  
   opening • 2-58  
     for appending • 2-59  
   outputting data to • 2-25  
 .SETF directive • 2-66  
 .SETN directive • 2-67  
 .SETS directive • 2-68  
 .SETT directive • 2-66  
 <SEVERE> symbol • 2-8  
 Special symbol • 1-3, 2-7  
   format • 2-7  
   logical • 2-7  
   numeric • 2-8  
   type • 2-7  
 Statistics mode • 2-31  
 String constant • 2-10  
 String expression • 2-11  
 String symbol • 2-7, 2-10  
   comparing • 2-36  
   defining • 2-21  
   setting • 2-68  
   testing • 2-41, 2-42

<STRLEN> symbol • 2-8  
 Subroutine  
   calling • 2-34  
   returning from • 2-64  
 Substitution mode • 1-2, 2-30  
 Substring  
   searching • 2-72  
 <SUCCESS> symbol • 2-8  
 Switches • 2-13  
 <SYDISK> symbol • 2-9  
 Symbol • 1-2  
   defining • 2-3  
   substituting • 2-12, 2-30  
   using • 2-6  
 Symbol name • 2-6  
 Symbol table • 2-3, 2-12  
 Symbol type • 2-6  
   defining • 2-7  
   logical • 2-6  
   numeric • 2-7  
   string • 2-7  
 <SYSTEM> symbol • 2-8  
 <SYSUIC> symbol • 2-9  
 <SYUNIT> symbol • 2-8

---

## T

---

Task  
   executing in parallel • 2-76  
   testing • 2-38, 2-39, 2-44  
 .TEST directive • 2-72  
 Text  
   displaying on terminal • 2-16, 2-18, 2-21  
 <TIME> symbol • 2-9  
 Time mode • 2-31  
 /TR switch • 2-13  
 <TRUE> symbol • 2-8  
 <TSKTSZ> symbol • 2-8

---

## U

---

<UIC> symbol • 2-9

---

## V

---

Variable

# Index

## Variable (Cont.)

testing • 2-72  
<VERSN> symbol • 2-9

---

## W

---

.WAIT directive • 2-75  
<WARNIN> symbol • 2-8

---

## X

---

.XQT directive • 2-76

## Reader's Comments

This form is for document comments only. Digital will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

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.

---

---

---

---

---

---

---

---

---

---

---

---

Please indicate the type of user/reader that you most nearly represent:

- Assembly language programmer
- Higher-level language programmer
- Occasional programmer (experienced)
- User with little programming experience
- Student programmer
- Other (please specify) \_\_\_\_\_

Name \_\_\_\_\_ Date \_\_\_\_\_

Organization \_\_\_\_\_

Street \_\_\_\_\_

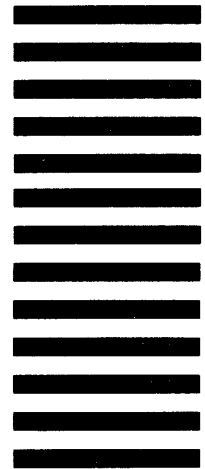
City \_\_\_\_\_ State \_\_\_\_\_ Zip Code \_\_\_\_\_  
or Country

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

IAS Engineering/Documentation  
Digital Equipment Corporation  
5 Wentworth Drive GSF/L20  
Hudson, NH 03051-4929



Do Not Tear - Fold Here