RT–11 IND Control Files Manual

Order Number AA–PDU1A–TC

August 1991

This manual tells you how to create and execute IND control files.

Revision/Update Information: This is a new manual for programmers; it is a revision of the IND chapter in the former *RT–11 System User's Guide*.

Operating System: RT–11 Version 5.6

Digital Equipment Corporation
Maynard, Massachusetts

First Printing, August 1991

The Reader's Comments form at the end of this document requests your critical evaluation to assist in preparing future documentation.

S1522

This document was prepared using VAX DOCUMENT, Version 1.2.

Contents

Tables

Preface

Document Structure

This manual is divided into the following three parts and appendix:

- Part I, Using IND Control Files, describes how to use the IND command language.
- Part II, Descriptions of IND Directives, alphabetically describes each IND directive.
- Part III, IND Error Messages, alphabetically lists and explains each error message.
- Appendix A, Sample IND Control Files, displays three sample control files.

Audience

This manual is written for those users of the RT–11 operating system who want to program control files.

Conventions

The following conventions are used in this guide.

Convention	Meaning

Black print	In examples, black print indicates output lines or prompting characters that the system displays. For example:
	`.BACKUP/INITIALIZE DL0:F*.FOR DU1:WRK`
	`Mount output volume in DU1:; continue? Y`

Red print	In examples, red print indicates user input.
Braces ({ })	In command syntax examples, braces enclose options that are mutually exclusive. You can choose only one option from the group of options that appears in braces.
Brackets ([ ])	Square brackets in a format line represent optional parameters, qualifiers, or values, unless specified otherwise.

Convention	Meaning
lowercase characters	In command syntax examples, lowercase characters represent elements of a command for which you supply a value. For example: `DELETE filespec`
UPPERCASE characters	In command syntax examples, uppercase characters represent elements of a command that should be entered exactly as given.
RET	RET in examples represents the RETURN key. Unless the manual indicates otherwise, terminate all commands or command strings by pressing RET.
RETURN	RETURN in the text represents the RETURN key.
CTRL/*x*	CTRL/*x* indicates a control-key sequence. While pressing the CTRL key, press another key. For example: CTRL/C.

Associated Documents

Basic Books

- *Introduction to RT–11*
- *Guide to RT–11 Documentation*
- *PDP–11 Keypad Editor User's Guide*
- *PDP–11 Keypad Editor Reference Card*
- *RT–11 Commands Manual*
- *RT–11 Mini-Reference Manual*
- *RT–11 Master Index*
- *RT–11 System Message Manual*
- *RT–11 System Release Notes*

Installation Specific Books

- *RT–11 Automatic Installation Guide*
- *RT–11 Installation Guide*
- *RT–11 System Generation Guide*

Programmer Oriented Books

- *RT–11 System Utilities Manual*
- *RT–11 System Macro Library Manual*
- *RT–11 System Subroutine Library Manual*
- *RT–11 System Internals Manual*
- *RT–11 Device Handlers Manual*
- *RT–11 Volume and File Formats Manual*
- *DBG–11 Symbolic Debugger User's Guide*

- The <JOBS>, <MAPPED>, and <SYSTEM> special numeric symbols have been updated to include the RT–11 5.6 monitors.
- A new special logical symbol <MODE> has been added to indicate if the current monitor supports mapping to Instruction and Data space.

See the *IND Reserved Symbols* section in Chapter 2 for a description of these symbols.

Part I
Using IND Control Files

Part I describes how to use the IND control-file processor to create and execute control files.

Chapter 1

Definition, Purpose, and Requirements

To understand control files, you should know what indirect files are and how control files are related to BATCH and command files.

Indirect Files

Indirect files consist of a series of commands that perform a task. These files allow you to process commands indirectly from a file rather than directly from the keyboard.

Indirect files are of three types:

- **BATCH**

  A file written in the BATCH job-control language, processed by the BATCH handler, and having a default BAT file type is called a batch file. See the *RT–11 System Utilities Manual* for how to create and use RT–11 BATCH files.

- **Command**

  A command file is written in the DCL command language, has a default COM file type, and is processed by KMON. See the *Introduction to RT–11* for how to create and use command files and see the *RT–11 Commands Manual* for DCL command descriptions.

- **Control**

  A control file is written in the IND control-file language, has a default COM file type, and is processed by the IND processor. This manual describes how to create and use IND control files.

RT–11 Control Files

RT–11 distributes the following control files:

- DATIME.COM (procedure for requesting the day and time)
- CONFIG.COM (configuration procedure)
- STRTAI.COM (automatic-installation procedure)
- SYSGEN.COM (system-generation procedure)
- VERIFY.COM (verification procedure for automatic installation)

A Sample Control File

The following control file describes how files backed up into savesets on tape are restored. This file illustrates the complexity of control files. The list following the example explains some of this file's contents.

```
.; ❶ RESALL.COM
.; Restores all files from all savesets on a tape.
.;
        .ASKS ❷ string Input device?
$ASSIGN ❸ 'string' ❹ BUD ❺
$LOAD 'string'
        .ASKS string Output device?
$ASSIGN 'string' OUT
        .SETS REWSWT "REWIND"          .; Start with a rewind
.100: ❻
$BACKUP/RESTORE/LOG/NOQUERY/'REWSWT' BUD:/SAVESET OUT:
        .SETS REWSWT "NOREWIND"
        .IF <EXSTAT> ❼ LT <ERROR>  .GOTO 100
;?RESALL-I-BACKUP exit status warrants termination of procedure. ❽
        .EXIT
```

❶ The period-semicolon combination (.;) specifies an internal comment. Anything on the line after them is comments for the programmer only and is not displayed when the COM file is run.

❷ The period-command combination (for example, `.ASKS`, `.SETS`, `.IF`, and `.EXIT`) specifies an IND directive or command.

❸ The dollar sign ($) followed by a DCL command specifies that command line is DCL. The dollar sign is optional, allowing an easy way for you to instantly identify a DCL command in a control file.

The IND processor forwards any line that does not begin with a period or a semicolon to KMON to interpret; and KMON disregards an initial dollar sign in a command line.

❹ The `'string'` specifies user input that the string will represent when the COM file is run.

❺ `BUD` is a logical device name meaning Back-Up Device.

❻ A label is an alphanumeric string preceded by a period and followed by a colon (.100:). Labels are local symbols specifying places in the control file so that the indicated lines can be referenced by an IND directive.

❼ `<EXSTAT>` and `<ERROR>` are IND special numeric symbols. This section of the example code means that if a saveset is successfully backed up and the tape does not rewind, then the next saveset is backed up.

❽ A line beginning with a semicolon specifies an external comment, which is displayed when the control file is run.

Command and Control Files Compared

Command Files

Command files are simpler than control files and can be called from control files. So, use a command file if you have a task that:

- Does not require user intervention
- Involves the processing of valid KMON commands that are complete on one line and are in sequential order

Control Files

You can use control files to interactively process information between a user and RT–11 or to automatically perform a task unassisted by a user. You can program control files to:

- Create, access, or execute other files
- Execute keyboard monitor commands
- Define symbols
- Pass parameters
- Perform logical tests
- Choose a path of action based on the results of a test

The Added Dimension of Control Files

If they are complete on one line, both command and control files can contain any valid KMON command, such as a:

DCL (Digital Command Language) command
CSI (Command String Interpreter) command
CCL (Concise Command Language) command
UCL (User Command Linkage) command
UCF (User Command First) command

In addition to KMON commands, control files also contain the added dimension of IND directives (commands).

See the *RT–11 Commands Manual* for an explanation of DCL commands, the *RT–11 System Utilities Manual* for an explanation of CSI and CCL commands, and the *Introduction to RT–11* for an explanation of UCL and UCF commands.

IND passes any control-file line not containing an IND directive to KMON to interpret.

Versatility of Control Files

RT–11 control files are more versatile than RT–11 DCL command files for the following reasons:

- Control files can ask questions with a prompt and get a reply. Command files cannot; they do not support user intervention once started.
- Control files can make logical tests, use symbol substitution, and transfer control to a subroutine or to another part of the file based on the results of the tests. Command files cannot; they do not support conditional logic and they process commands only in sequential order.
- Control files have all the capabilities of command files plus the features of the IND control language.

For example, in the following command file, you can initialize disk DZ1, copy all the files from disk DL0 to DZ1, and get a directory listing to check the copy procedure:

```
INITIALIZE/NOQUERY DZ1:
COPY/SYSTEM DL0:*.*  DZ1:*.*
DIRECTORY DZ1:
```

However, you can use this command file for only this one situation. You have to write a new command file when you want to copy files to or from a different disk.

On the other hand, you can write one control file to fit all disks. The following example control file interactively asks what devices you want to use. The user is to interactively supply the device names, which IND respectively stores in the symbols INDEV and OUTDEV when the control file is run:

```
.ASKS INDEV From what device do you want to copy?
.ASKS OUTDEV To what device do you want to copy?
$INITIALIZE/NOQUERY 'OUTDEV'    .;This will be processed by KMON.
$COPY/SYSTEM 'INDEV' 'OUTDEV'   .;This will be processed by KMON.
$DIRECTORY 'OUTDEV'             .;This will be processed by KMON.
```

You can also:

- Set up a table
- Do a logical test of a device based on information in the table
- Copy files to the correct type of device based on the results of the logical test

This procedure can be done in a control file without your having to respond interactively.

Requirements for Using IND

To run IND, you must have:

- The file IND.SAV (the IND processor) on your system device.

  If IND.SAV is not on your system device, you must specify the device where IND is located. You can, however, do a software customization to change the default device from which IND.SAV is run; see customization 39 in the *RT–11 Installation Guide* for instructions on how to do this.

- Approximately 12K contiguous words of available low memory.

  Because IND demands so much low-memory space, you may need to unload some jobs or device handlers to run IND.

How to Make Low-Memory Space Available to Run IND

If you need more memory space to run IND, do the following:

1. Issue a SHOW MEMORY command to see how much available low-memory space you have and what jobs or device handlers are currently loaded.

   The available low memory is given in the last line of the SHOW MEMORY display, under the region name ..BG...

2. Under a multi-job monitor, stop and unload unnecessary jobs. Look at the low-memory space taken by your jobs and, if necessary, decide which ones to unload to give you the needed low memory. For example, the following two steps stop and unload SPOOL and VTCOM:

   a. Stop the job:

   ```
   SET SP EXIT to stop SPOOL
   CTRL/P and EXIT to stop VTCOM
   ```

   b. Unload the job:

   ```
   UNLOAD SPOOL to unload SPOOL
   UNLOAD VTCOM to unload VTCOM
   ```

3. Under any monitor, unload and remove unnecessary device handlers. Look at the low-memory space taken by your device handlers and, if necessary, decide which ones you can unload. For example, the following commands unload and deassign the device handlers for the SPOOL and VTCOM jobs:

   ```
   UNLOAD SP      for SPOOL
   UNLOAD LS
   DEASSIGN LP
   DEASSIGN LP0

   UNLOAD XL      for VTCOM
   ```

4. Make sure with the SHOW MEMORY command that you have gathered all the free low memory in one place under the region name ..BG... You cannot run IND if the needed low memory is not consolidated.

Note: To prevent fragmentation of low-memory space, device handlers and jobs must be unloaded in reverse order from that in which they were loaded.

See the Using System Jobs chapter in the *Introduction to RT–11* for some general ways of stopping jobs and unloading utilities.

5. Run IND.
6. Install (reload) the jobs and handlers you unloaded, as needed.

Using an Edited STRTxx.COM File to Set Up the Required Space

To get the correct amount of memory, reboot your operating system with an edited STRTxx.COM file. When you reboot your system, the desired amount of low memory will be available:

- Copy your STRTxx.COM to STRTxx.DIS.
- Edit your STRTxx.COM to load only the utilities that will leave you adequate low-memory space for running IND.
- Reboot your operating system.
- Run IND.
- When your control file is complete, copy the STRTxx.DIS to STRTxx.COM.
- Finally, reboot your operating system.

Chapter 2

Creating a Control File

Use a text editor to create a control file. Call the file by any name you wish, but give it a file type of COM since this extension is the default used by IND to locate the file.

Choose a name for the control file that suggests the task the file performs. Preface the commands in the file with a comment header containing the name, creation and edit dates, and a short description of the file's function. End the last command line of the control file with a RETURN if you want that line to be processed by IND.

Control-File Line

The following is the general format of a control-file line.

Format

```
{ [ [$]KMON_command ]                                                }
{ [ .label ]  [ IND_directive ]  [ symbol[s] ]  [ [.];comment ]      }
```

Components

$KMON_command

Executes a task. You can issue any valid KMON command in a control file as long as it is complete on one line. Valid KMON commands include DCL, CCL, CSI commands, or commands you define using UCL or UCF. See the *RT–11 Commands Manual* for a description of KMON commands.

.label

Assigns a name to a line in the control file so the line can be referenced.

IND_directive

Executes a task.

symbol[s]

Functions as a variable allowing you to control the flow of execution.

[.];comment

Explains what the control file does. The comment line is displayed when the control file is run if the period preceding the semicolon is omitted.

Description

An IND control-file line can consist of a valid KMON command, code unique to IND, or a combination of both.

IND attempts to interpret all lines beginning with a period or a semicolon. IND passes all other lines and anything it does not recognize to KMON to interpret.

You must precede each IND directive with a period (.). You can enter keyboard commands without a preceding character, but Digital recommends that you precede those commands with a dollar sign ($) to distinguish them from IND directives.

You can issue IND directives and DCL or CCL commands together on the same line or on separate lines. You can also issue multiple directives on the same line.

However, you cannot issue an IND directive on a command line after a KMON command on the same line. Once IND passes KMON a command line or a section of a command line, IND does not receive that command line back. If KMON cannot interpret everything passed to it on the command line, it displays an error message.

Example

The following is a sample line of code from a control file with an explanation of the line's components:

```
.START:❶ .ASKS❷ ANS❸ Do you want a printed copy?  .;This determines❹
                                                   .;where the results
                                                   .;of the procedure
                                                   .;will be displayed.
```

❶ A label

❷ A directive that displays the prompt `Do you want a printed copy?`

❸ A symbol that stores the answer the user gives to the prompt

❹ An internal comment intended for the programmer

The following sections of this chapter beginning with *Labels* and ending with the *Chapter Summary* describe how to use the components of a control-file line to create control files.

Labels

Defining Labels

A label is defined at the *beginning* of a control line. When defined, a label consists of a period (.), an alphanumeric string of up to six characters, and a colon (:). A label may also contain a leading, trailing, or embedded dollar sign ($).

There are two types of labels: direct access and nondirect access. A direct-access label is on a line by itself. A nondirect-access label also has a directive, a DCL command, or a comment.

Referencing Labels

By referencing labels with conditional expressions, you can direct the processing of your control file to different sections or subroutines. The following excerpt from the annotated sample control file in Chapter 1 shows how a label is both defined and referenced:

```
.if <SYSTEM>  EQ 7 .goto 900   .; If SB, quit.
   .
   .
   .
.900:
```

Note that when a label is referenced in an argument of a directive, it contains no punctuation.

See the description of the .label directive for more information on using labels. See the descriptions of the following directives for other examples of the use of labels:

.GOSUB
.GOTO
.IF
.IFDF/.IFNDF
.IFENABLED/.IFDISABLED
.IFLOA/.IFNLOA
.IFT/.IFF
.ONERR
.TEST

Commands

You can issue both IND directives and valid KMON commands in a control file.

The following sections describe how to issue IND directives and DCL, CCL, and CSI commands from within a control file.

Issuing IND Directives

IND directives are commands that perform the following functions:

- Define and assign values to three types of symbols: logical, numeric, and string
- Determine file structure
- Create and access data files
- Control the logical flow in a control file
- Perform logical tests
- Enable or disable operating modes
- Increment or decrement a numeric symbol
- Perform operations driven on time intervals, if you have timer support

You can issue directives on a line with a label or a comment or alone on a line. If you use them with labels or comments, you must place directives after a label but before a comment, as in the following example code:

```
LABEL           DIRECTIVE                       COMMENT

.DEVICE:        .IFLOA MU: .GOSUB COPY          .;If MU: is loaded,
                                                .;go to COPY subroutine.
```

IND directives are described in detail in Part II.

Operating Modes

An operating mode is the way one or more directives can be interpreted by IND. You can change operating modes through the .ENABLE and .DISABLE directives. The following table lists the IND operating modes:

```
ABORT        DELETE       MCR         SUBSTITUTION    TYPEAHEAD
CONTROL-Z    ESCAPE       OCTAL       SUFFIX
DATA         GLOBAL       PREFIX      TIMEOUT
DCL          LOWERCASE    QUIET       TRACE
```

Each mode is independent of the others, and all can be active simultaneously. See the description of the .ENABLE directive for individual descriptions of each mode.

Issuing DCL Commands in Control Files

You can issue all DCL commands, except those noted below, in control files as long as they are complete on one line. These commands are described in detail in the *RT–11 Commands Manual*.

DCL commands can appear with labels, but not on lines with the IND comment character (;). When IND encounters something it does not recognize, IND passes that information along with the remainder of the command line to the keyboard monitor. Since the keyboard monitor does not interpret the IND comment, an invalid command error results. However, in this situation, you can use the KMON comment character (!).

If you use keyboard commands that query (such as the DCL DELETE command with wildcards) or prompt you (such as the DCL LINK/PROMPT command), you must either specify the /NOQUERY option or enter the response to the query at the terminal when the command is executed.

NOTE

The following DCL commands should not be used in control files, as they will produce unpredictable results:

```
B(ase)              REENTERR
D(eposit)           SAVE
E(xamine)           START
GET
```

Issuing CCL Commands in Control Files

You can issue CCL commands in control files to run the utility programs described in the *RT–11 System Utilities Manual*. The following example executes a CCL command if the RL01/RL02 device handler is loaded:

```
.ROUTN:  .IFLOA DL PIP DL1:MYPROG.SAV=DL0:MYPROG.SAV
```

The following command line copies a diskette in image mode if a certain condition is true. The .IFT directive tests the condition represented by the symbol LOGSYM for a true or false value:

```
.IFT LOGSYM DUP DU1:*=DU0:/I/Y
```

If manual intervention is not desired in a control file, issue the commands with options to prevent querying (as does /Y in the preceding example). Otherwise, commands that require user intervention will prompt for user input when the control file is executed.

Issuing CSI Commands Indirectly from a Control File

Since each command in a control file must be complete on one line, you cannot directly issue CSI commands in a control file, unless they are in CCL form. However, you can process commands with more than one line by using the .ENABLE DATA or .DATA directive to create and execute a command file from within the control file.

The control-file lines in the next two examples create and execute a command file that runs PIP, unprotects the file DY:FILE.TST, and copies the file to DY1.

The first example uses the .ENABLE DATA directive to create the command file:

```
.ASKS DEV WHICH DEVICE CONTAINS FILE.TST?
.OPEN COPY.TMP
.ENABLE DATA
RUN PIP
'DEV':FILE.TST/Z
DY1:FILE.TST='DEV':FILE.TST/W/Y
^C
.DISABLE DATA
.CLOSE
$@COPY.TMP
```

The second example uses the .DATA directive to create and execute the same command file:

```
.ASKS DEV WHICH DEVICE CONTAINS FILE.TST?
.OPEN COPY.TMP
.DATA RUN PIP
.DATA 'DEV':FILE.TST/Z
.DATA DY1:FILE.TST='DEV':FILE.TST/W/Y
.DATA ^C
.CLOSE
$@COPY.TMP
```

See Part II for information on using the .ENABLE DATA and .DATA directives.

Symbols

An IND symbol is a name that represents a logical value, a number, a character, or a group of characters. That name can consist of one to six alphanumeric characters that can include one or more dollar signs ($).

Symbol Types

IND symbols are classified according to the type of data they store:

- **Logical**

  Symbols that have either a true or false value.

- **Numeric**

  Symbols that have an integer value.

- **String**

  Symbols that have an ASCII character or a string of ASCII characters as a value.

Symbol Uses

You can use IND symbols in much the same way as you use symbols in most programming languages. You can use these symbols in your control files as constants or variables to represent logical, numeric, or string values. By testing or comparing the symbols, you can control what your control file will do.

You can substitute a symbol for a:

- User's response to a question
- File name
- Number
- True or false value
- DCL command
- Data records
- Data files
- Comments to be displayed at the terminal

Logical Symbols

A logical symbol represents either TRUE or FALSE. These symbols are useful when you need a YES or NO response to a control-file question. For example, consider the following IND control-file code:

```
.ASK DONE Are you finished
.IFT DONE .GOTO BYE
.IFF DONE .GOTO OK
.BYE:     ;So long.
.EXIT
.OK:      ;What else is new?
.EXIT
```

The preceding code displays the following prompt on your terminal:

```
* Are you finished? [Y/N D:N]:
```

The IND directive .ASK expects a Y (YES) or an N (NO) as a response and assumes NO as the default answer (if you only press RETURN in response to the prompt). This is indicated by the `[Y/N D:N]` in the prompt.

In the preceding code sample, IND stores the Y or N in the logical symbol DONE:

- If DONE contains Y, IND displays the message:

  ```
  ;So long.
  ```

- If DONE contains N, IND displays the message:

  ```
  ;What else is new?
  ```

You define, redefine, and use logical symbols through the .ASK, .SETF, .SETL, and .SETT directives. See the descriptions of these directives in Part II for further information.

Numeric Symbols

A numeric symbol represents an integer. These symbols are useful when you want to use numeric variables in a control file. For example, you can use these symbols to:

- Represent integer arguments in command lines.
- Combine the integers the symbols represent with other numeric symbols and constants to form arithmetic expressions.

Range

The valid range for an integer represented by a numeric symbol is 0 to 65535_{10} or 177777_8.

Radix

A numeric symbol can have either an octal or decimal radix. The default is octal. The following determines the radix:

- **Octal**
  - When OCTAL is enabled (.ENABLE OCTAL), which is the default, the value of the symbol is stored as an octal number (unless you specify it to be decimal).
  - You can specify a number to be octal (whether or not OCTAL is enabled) by placing a number sign (#) before that number. For example:

    ```
    #532
    ```

    Note that when you specify a number to be octal while OCTAL is disabled, that octal number is stored in its decimal equivalent.
  - IND displays an (O) when it expects you to respond to a prompt with an octal number.
- **Decimal**
  - When OCTAL is disabled (.DISABLE OCTAL), the value of the symbol is stored as a decimal number (unless you specify it to be octal).
  - You can specify a number to be decimal (whether or not OCTAL is enabled) by placing a decimal point (.) after the number. For example:

    ```
    532.
    ```

    Note that when you specify a number to be decimal while OCTAL is enabled, that decimal number is stored in its octal equivalent.
  - IND displays a (D) when it expects you to respond to a prompt with a decimal number.

You can change a symbol's radix by using the .SETD or .SETO directive. See the descriptions of these directives in Part II.

Example

```
.DISABLE OCTAL
.ASKN NUMBER How many people are you expecting for dinner
.IF NUMBER GT 5 .GOTO  STORE
.IF NUMBER LE 5 .GOTO  HOME
.STORE:    ;Go buy some more wine.
.EXIT
.HOME:     ;Please set 'NUMBER' extra places at the dining room table.
.EXIT
```

Next are two examples of user responses to the preceding code when it is run:

```
* How many people are you expecting for dinner [D]: 9
;Go buy some more wine.
@ <EOF>
```

and

```
* How many people are you expecting for dinner [D]: 2
;Please set 2 extra places at the dining room table.
@ <EOF>
```

You define, redefine, and use numeric symbols through the .ASKN, .DEC, .INC, .SETD, .SETN, and .SETO directives. See the descriptions of these directives in Part II for further information.

Numeric Expressions

You specify numeric expressions by using operators to combine numeric symbols and constants. Table 2–1 lists the operators you can use.

Table 2–1: Operators for Numeric Expressions

Operator	Function
Arithmetic	
+	Add
–	Subtract
*	Multiply
/	Divide
( )	Subexpressions
Logical	
!	Logical inclusive OR
&	Logical AND tests
^	Logical NOT tests
( )	Subexpressions
Relational	
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
( )	Subexpressions

Two Ways IND Handles Numeric Expressions

- When you divide numeric symbols, IND always truncates the dividend to yield an integer.
- Numeric expressions are evaluated from left to right, unless you use parentheses to form subexpressions, which IND evaluates first. There is no operator precedence. The following code displays the line `;N3 equals 20 (decimal)`:

```
.DISABLE OCTAL
.SETN N1 2
.SETN N2 3
.SETN N3 N1+N2*4
;N3 equals ’N3’ (decimal)
```

In the next example, IND assigns the symbol N3 the value 16_{10} and displays the line `;N3 equals 16 (decimal)`:

```
.SETN N1 2.
.SETN N2 3.
.SETN N3 N1+(N2*4)
;N3 equals ’N3’ (decimal)
```

Radix of Numeric Expressions

- **Octal Radix**

  If octal mode is enabled and you specify no decimal values, the expression is evaluated and the result is stored as an octal value.

- **Decimal Radix**

  If octal mode is enabled and you use both octal and decimal values in an expression, all octal values are converted to decimal before the expression is evaluated and the result is stored as a decimal value.

  If octal mode is disabled or you use all decimal values in an expression, the expression is evaluated and the result is stored as a decimal value.

String Symbols

A string symbol represents an ASCII character or a string of up to 132 ASCII characters.

String symbols can save a user's responses to questions. Responses stored in string symbols let you plan different ways to execute the control file, depending on a user's situation.

Assigning Values to String Symbols

You can make the following assignments to string symbols:

- **Character string**

  To assign a character string to a string symbol, enclose the character string with quotes, as in the following example, which assigns a string to the string symbol PROMPT:

  ```
  .SETS PROMPT "Do you have any protected files"
  ```

- **String symbol**

  To assign a string symbol to another string symbol, issue the .SETS directive. In the following example, the string symbol NAME is assigned the contents of a previously defined string symbol, VOLID. NAME becomes a copy of VOLID, making both symbols identical until you change them:

  ```
  .SETS NAME VOLID
  ```

- **Empty string**

  To assign an empty string to a string symbol, specify the string with two double quotes. The empty string specifies the NULL character to IND. In the following example containing an empty string, if the user just presses RETURN in response to the question, IND sets the time to zero:

  ```
  .ASKS TIME Time to display the announcement (hh:mm)?
  .IF TIME EQ "" .SETS TIME "0"
  ```

Extracting Substrings from String Symbols

IND lets you extract substrings from string symbols. In the following example, string symbol MYDEV is assigned four characters from the string represented by DEVS:

```
.SETS DEVS "DEVICE DU0: DU1: MU0:"
.SETS MYDEV DEVS[8.:11.]
```

The [8.:11.] indicates that characters 8 to 11 (decimal) of the string symbol DEVS are to make up the string symbol MYDEV. MYDEV consequently contains the characters DU0:. The square brackets are part of the command line; you must use them when you specify the range for a substring. If you use a decimal point with either number in the range specification (that is, the numbers that appear between the brackets), IND interprets both numbers as decimal.

You can use substrings only with the .SETS and .IF directives and only as second arguments to those directives.

Adding Strings to String Symbols

You can concatenate string symbols with other string symbols, substrings, and character strings by using the plus sign (+). For example, the following directive concatenates the string JANFEBMARAPRMAYJUNJULAUGSEPNOV to the string symbol DEC and assigns the combined value to string symbol YEAR:

```
.SETS YEAR JANFEBMARAPRMAYJUNJULAUGSEPNOV+"DEC"
```

See the description of the .SETS directive in Part II for concatenating string symbols.

You define (assign), redefine, and use string directives through the .ASKS, .READ, and .SETS directives. See the descriptions of these directives in Part II for further information.

Substituting the Values of Symbols for the Symbols

By default IND replaces a symbol in a command line with its value. This mode of operation is called substitution mode. You can change the mode through the .DISABLE/.ENABLE SUBSTITUTION directives. You can use symbol substitution with logical, numeric, or string symbols. Using substitution mode you can:

- Display character strings and prompts at the terminal.
- Manipulate symbol values neatly and efficiently in your control files.
- Substitute the values of symbols in comments that IND displays at the terminal.

Specifying Substitution

You must enclose a symbol with apostrophes if you want IND to substitute a value for it. If IND encounters an apostrophe when substitution is enabled, IND treats the subsequent text, up to a second apostrophe, as a symbol name. IND then substitutes the value of the symbol in the command line in place of the symbol. Note the following example control-file code:

```
.ENABLE SUBSTITUTION
.ASKS DEV ENTER INPUT DEVICE
$ASSIGN 'DEV' INP
```

When IND processes the preceding lines, it displays the following:

```
* ENTER INPUT DEVICE [S]:
```

Suppose the user responds to the displayed prompt with the string DU1. This response would assign the string value DU1 to the string symbol DEV, and IND would have translated the next line of code:

```
$ASSIGN 'DEV' INP
```

to

```
$ASSIGN DU1 INP
```

If substitution mode is disabled, IND passes the line to KMON as it appears in the control file (that is, `ASSIGN 'DEV' INP`) instead of `ASSIGN DU1 INP`.

Including Apostrophes As Text in a Displayed Comment

If substitution mode is enabled, an apostrophe signals the beginning of a string symbol. To include an apostrophe as text in a displayed comment, rather than as the start of a symbol, you must replace the single apostrophe with two consecutive apostrophes (`''`). For example, if substitution mode is enabled, IND displays the control-file line:

```
;THE SYMBOL''S VALUE
```

as

```
;THE SYMBOL'S VALUE
```

IND Reserved Symbols

IND has a set of reserved symbols also called *special* symbols that it defines according to specific system characteristics and responses to queries during command execution. Like the symbols you define, these symbols can be of three types: logical, numeric, and string.

You cannot define or change reserved symbols, though you can use them and test them to determine system characteristics.

IND special symbols are enclosed in angle brackets so you can distinguish them from symbols you create. Table 2–2 lists these symbols.

Table 2–2: IND Special Symbols

Symbol	Value
Logical Symbols	
<ALPHAN>	Set to true if last string entered in response to an .ASKS directive contains only alphanumeric characters. The .TEST and .TESTFILE directives can set <ALPHAN> to true or false. An empty string sets <ALPHAN> to true, and any embedded blank sets <ALPHAN> to false. If lowercase mode is disabled, a lowercase response to a query sets <ALPHAN> to false.
<ALTMOD> or <ESCAPE>	Set to true if the last query was answered with a single escape character. Otherwise set to false.
<DEFAUL>	Set to true if response to last query was a default response (that is, a carriage return was entered).
<EOF>	Set to true after the .READ directive encounters end-of-file. Otherwise set to false.
<FALSE>	Permanently set to false; can be used to specify a default response for an .ASK directive.
<MAPPED>	Set to true if IND is running under a mapped monitor; false if otherwise.
<MODE>	Set to true if the monitor is an RT–11 ZB or ZM monitor, which means the monitor supports the mapping of Instruction and Data space. Set to false if the monitor is an RT–11 FB, SB, XB, or XM monitor, which means the monitor does not support the mapping of Instruction and Data space.
<OCTAL>	Set to true if the last numeric symbol tested with the .TEST directive contained an octal value; false if the last numeric symbol contained a decimal value. Also set to true if a numeric symbol defined using the .SETN or .ASKN directive was assigned an octal value; false if assigned a decimal value.

Table 2–2 (Cont.): IND Special Symbols

Symbol	Value
<RAD50>	Set to true if last string entered in response to an .ASKS directive, or tested with a .TEST or .TESTFILE directive, contains only Radix–50 characters. The period and dollar sign are valid Radix–50 characters, but blank characters and lowercase alphabetic characters are not. An empty string sets <RAD50> to true.
<TIMOUT>	Set to true if the last response to an .ASK directive exceeded the timeout count; false if otherwise.
<TRUE>	Permanently set to true; can be used to specify a default response for an .ASK directive.
Numeric Symbols	
<ERROR>	Permanently assigned the value 2 (octal) to represent an error.

<EXSTAT>	Assigned the exit status value of 0, 1, 2, or 4 according to the contents of the user error byte as a result of the last keyboard command executed. The values 0, 1, 2, and 4 indicate:
	`0  Warning`
	`1  Success`
	`2  Error`
	`4  Fatal error`
	This reserved numeric symbol is modified at the completion of each keyboard command. The .EXIT directive can also modify <EXSTAT>. The value is returned from the task that has completed if you have used the .EXIT directive to specify an exit status value. Otherwise, the value is returned from KMON.

<FILERR>	Assigned a numeric status code indicating whether a file operation was successful. The codes returned, along with their descriptions, are:
	`377  Read-related error`
	`372  No room to fetch handler`
	`366  End-of-file detected`
	`363  Data overrun`
	`351  Device full`
	`346  No such file`
	`343  File accessed for write`
	`340  Device read error`
	`337  Device write error`
	`333  No file accessed on channel`
	`327  File exceeds space allocated`
	`325  Bad record type (non-ASCII)`
	`324  File accessed for read`
	`313  File already open`
	`312  Bad file name`
	`244  Invalid device or unit`

Table 2–2 (Cont.): IND Special Symbols

Symbol	Value
<JOBS>	Assigned the octal number of jobs that the running monitor can support. The SB, XB, and ZB monitors support 1 job; the FB monitor can support 2 jobs, and the XM and ZM monitors can support up to 8_{10} (10_8) jobs.
<SEVERE>	Permanently assigned the value 4 (octal) to represent a fatal error.
<SPACE>	Assigned an octal number to represent the amount of space available, in bytes, in the symbol table. IND uses a minimum of 10 bytes (octal) for each symbol.
<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.
<SUCCES>	Permanently assigned the value 1 (octal) to represent success.

<SYMTYP>	Assigned a numeric code to represent the type of symbol tested by a .TEST directive. The codes represent the following:
	`0  Logical symbol`
	`2  Numeric symbol`
	`4  String symbol`

<SYSTEM>	Assigned an octal number to represent the operating system on which IND is running. The octal numbers represent one of the following:
	`0   RSX-11D`
	`1   RSX-11M`
	`2   RSX-11S`
	`3   IAS`
	`4   RSTS`
	`5   VAX/VMS`
	`6   RSX-11M-PLUS`
	`7   RT-11`
	`    (DCL command SET MODE SJ has been executed)`
	`10  RT-11`
	`    (DCL command SET MODE SJ is not in effect)`

Table 2–2 (Cont.): IND Special Symbols

Symbol	Value
	You can identify an RT–11 V5.6 monitor according to the following values of the <MAPPED>, <MODE>, and <JOBS> reserved symbols.

Monitor	<MAPPED>	<MODE>	<JOBS>
FB	F	F	up to 2_{10} (2_8) jobs
SB	F	F	1
XB	T	F	1
ZB	T	T	1
XM	T	F	up to 8_{10} (10_8) jobs
ZM	T	T	up to 8_{10} (10_8) jobs

If <JOBS> is more than 1, the FB, XM, or ZM monitor is running. If <MAPPED> is true, the XB, XM, ZB, or ZM monitor is running. If <MODE> is true, the ZB or ZM monitor is running.

<SYUNIT>	Assigned the unit number of the user's system device (SY:).
<WARNIN>	Permanently assigned the value 0 (octal) to represent a warning.

String Symbols

<DATE> Assigned the current date; format is dd-mmm-yy. The length of the value of <DATE> is predetermined (nine characters). If date is defined with less than nine characters, the value is padded with blank characters in the symbol table. If no date has been assigned, <DATE> contains nine blank characters.

<EXSTRI> Assigned the physical device name, device size, and attributes of a device tested with the .TESTDEVICE directive. The following mnemonics are found in <EXSTRI>:

```
NSD   No such device
LOD   Device handler loaded
UNL   Device handler not loaded
ONL   On line
OFL   Off line
UNK   Unknown
MTD   Mounted
NMT   Not mounted
ATT   Attached
NAT   Not attached
```

For word positions, see .TESTDEVICE description in Part II.

Table 2–2 (Cont.): IND Special Symbols

Symbol	Value
<FILSPC>	Assigned the full file specification (device, file name, and file type) of the file tested with the .TESTFILE directive, opened with the .OPEN, .OPENA, or .OPENR directive, or opened when a control file is called to begin execution.
<MONNAM>	Assigned the name of the currently running monitor. The length of the value of this reserved string symbol is predetermined (six characters). Therefore, if <MONNAM> is defined with less than six characters, the value is padded with blank characters in the symbol table.
<SYDISK>	Assigned the device mnemonic of the user's system device.
<TIME>	Assigned the current time; format is hh:mm:ss.

Scope of IND Symbols

Symbols can be local or global:

- **Local**

  Local symbols are recognized only within the begin-end block or control file in which they are defined.

  All user-defined symbols and the reserved command-line symbols are local by default.

- **Global**

  Global symbols are recognized not only in the control file in which they are defined but also in any nested control files; that is, in control files called from the file in which the global symbols are defined. Global symbols are also recognized through all levels of nested control files.

  All reserved special symbols are global by default; they are always available to any command string in a control file. Note, however, that the IND .DUMP directive lists special symbols in a category separate from global symbols. The .DUMP directive classifies as global only user-defined symbols.

  You can define a symbol to be global by doing all of the following:

  1. Issuing the .ENABLE GLOBAL directive
  2. Placing the symbol on a line that executes after the .ENABLE GLOBAL directive is executed
  3. Beginning the symbol with a dollar sign ($)

Symbols are local if:

- The control file containing them has no .ENABLE GLOBAL directive.
- They begin with a dollar sign but precede the execution of an .ENABLE GLOBAL directive.
- They follow the execution of an .ENABLE GLOBAL directive but do not begin with a dollar sign.

Once a symbol is defined as local or global, you cannot change its scope. For example:

- If a symbol that begins with a dollar sign ($) is defined before the .ENABLE GLOBAL directive is executed, that symbol is defined as local and remains local for all the control-file execution.
- After an .ENABLE GLOBAL directive is executed, if another symbol with the same name is defined, there will be two symbols with the same name: one local symbol and one global symbol.

IND Command-Line Symbols

IND saves all the components of your command line in reserved command-line symbols. They are all string symbols with the exception of the last, which is numeric.

Symbol	Description
COMMAN	Stores the complete command with any file specifications, options, and parameters. By using the COMMAN symbol, you can set up a control file to parse its own command line.
P0	Stores the control-file specification and any options.
P1 P2 P3 P4 P5 P6 P7 P8 P9	Symbols P1 through P9 store parameters passed to the control file. P1 contains the first space-delimited parameter, P2 the second, and so on. P9 contains any remaining parameters. If you have less than 9 parameters, IND assigns a null value to the remaining string symbols.
<STRLEN>	Stores the file specification plus the number of P symbols set, excluding the symbols that contain null values. With this symbol you can check the number of parameters used when the control file is executed.

See the *Passing Parameters to a Control File* section in Chapter 3 for how to use command-line symbols. The last example of that section has the command line: `ADD 24 25`. IND stores this command line in the following way. Note that the full file specification is ADD.COM. The COM file type is not necessary since it is the default.

Symbol	Value
COMMAN	ADD 24 25
P0	ADD
P1	24
P2	25
P3	null
.	.
.	.
.	.
P9	null
<STRLEN>	3

Special-Function Characters

IND special-function characters are IND symbols that tell the IND processor how to interpret information that follows them. Table 2–3 lists these characters with their functions.

Table 2–3: IND Special-Function Characters

Character(s)	Function
.	Used as a prefix character for all IND directives and labels, *and* as a suffix character for all integers that have a decimal radix.
#	Used as a prefix character to define a response to an .ASKN directive prompt as octal. *Also* used as a prefix character for file numbers specified with the .CLOSE, .DATA, .OPEN, .OPENA, .OPENR, and .PURGE directives.
@	Used before a nested control-file specification to pass control to that file.
$	Used as a prefix character to define symbols as global.
$@	Used before a command-file specification to pass control to that file.
/	Used as an end-of-file character. This character has the same effect as the .STOP directive. See the description of both in Part II for more details.
+	Concatenates string symbols (see the *String Symbols* section).
;	Indicates the start of a comment that will be displayed during control-file execution. Comments can have up to 132 characters.
.;	Indicates the start of an internal comment, one that will not be displayed during control-file execution.

Comments

You can document a control file by including comments. Comments are useful since they not only provide information to a programmer but also provide directions to the user as the procedure executes.

You can include either external or internal comments in a control file:

- **External Comments**

  During execution, IND displays external comments at the terminal. To define an external comment, begin the comment with a semicolon (;).

  You can place an external comment on a line with a label and with directives that do not branch to another location. An external comment on a line with a branching directive is never displayed, because IND branches before the comment is processed.

  The following line shows an external comment on a line with a label and directives:

  ```
  .DEVICE: .IFLOA MU: .GOSUB COPY  ;Testing to see if MU is loaded.
  ```

- **Internal Comments**

  Internal comments document a control file internally and are never displayed at the terminal. To define an internal comment, begin the comment with a period and semicolon (.;).

  You can place an internal comment on a line with a label or with most directives. When used on a line with a label or with directives, the comment should be the last field on the line.

  The following line shows an internal comment. The second half of the comment appears on a line by itself:

  ```
  .DEVICE: .IFLOA MU: .GOSUB COPY         .;MU: is loaded, go to
                                          .;COPY subroutine
  ```

The length of the comment displayed depends on the capability of the terminal to display various line lengths. On terminals set up for 80 columns, comments can contain up to 80 characters including the period, semicolon, return, and line-feed characters. On terminals set up for 132 columns, comments can contain up to 132 characters including the period, semicolon, return, and line-feed characters.

NOTE

You cannot place an IND comment (internal or external) on a line after a DCL or CCL command, since the keyboard monitor will interpret the comment as an invalid command. However, you can place a DCL comment character (!) after a DCL command.

Displaying/Printing Nonprintable Characters

IND allows the display of nonprintable characters in a printable format. The conversion applies only between character and numeric symbols (not to logical symbols) and does not require the .ENABLE SUBSTITUTION directive.

You can convert and substitute the low-order byte of a specified numeric symbol to its equivalent ASCII character. Conversely, you can convert and substitute the first letter of an ASCII string symbol to an octal integer value representing that letter's position in the ASCII collating sequence.

The format of the string to be converted is:

'symbol%V'

where:

'	Is the single-quote character that must enclose the string.
symbol	Is the predefined string or numeric symbol to be converted and substituted.
%V	Specifies a conversion to the other type of format.

For example:

```
.SETN  ESCVAL 27.
.SETS  ESC "'ESCVAL%V'" .;Make the ESCAPE character into a string symbol
.SETS  HOME "'ESC'[0H"
.SETS  CLEAR "'ESC'[0J"
; 'HOME''CLEAR'         .;Clear the screen ...
; Escape = 'ESC%V'
.SETS   S "A"           .;Define a STRING symbol
.SETN   N 'S%V'         .;Convert STRING character to a NUMBER
; Value of 'S' = 'N'
.SETS   S "'N%V'"       .;Convert NUMBER to STRING
; Character 'N' = 'S'
```

The preceding code produces a cleared screen and then displays:

```
         .;Clear the screen ...
; Escape = 33
; Value of A = 101
; Character 101 = A
@ <EOF>
```

Opening Devices for Read or Write Operations

IND provides support for read and write operations on devices. IND allows the opening of any device type for write operations (.OPEN directive) and the opening of any directory-structured device for read operations (.OPENR directive). IND provides this support for devices such as LP, LS, SP, and the file-structured magtape handlers.

This support, along with support for .TESTDEVICE and .VOL on magtape devices, is the only support IND provides for special directory-structured devices. However, this support does allow access to such devices from IND control files.

Some IND directives are inappropriate for magtape devices because magtapes are serial devices. For example, you cannot issue the .OPENA directive to a magtape device. And, also, you can open only one file at a time on a magtape volume.

Chapter Summary

Components of the IND Programming Language

The IND programming language includes the following components:

Directives

Commands that direct the execution of control files. Directives specify actions.

Operating Modes

Different ways directives can function.

Logical Symbols

Symbols you define or redefine to represent true or false values.

Numeric Symbols

Symbols you define or redefine to represent integers.

Arithmetic, Logical, and Relational Operators

Characters that allow you to form numeric, logical, and relational expressions and tell IND how to perform arithmetic and logical operations.

String Symbols

Symbols you define or redefine to represent a character or a string of characters.

Reserved Symbols

IND-defined symbols that IND sets according to specific system characteristics and responses to queries during command execution. You cannot set them directly, though you can use them.

Command-Line Symbols

IND-defined string symbols in which IND saves the components of your IND command line.

Special-Function Characters

Characters that tell the IND processor how to interpret information that follows them (as directives, comments, and so on).

Symbol Attributes

IND symbols have the following attributes:

- Origin (user-defined, reserved—special and command-line)
- Scope (local, global)
- Type (logical, numeric, string)

Syntax Rules for Creating Control Files

In general, follow these rules of syntax when you create control files:

1. **Directive/Commands**

   a. Precede IND directives with a period (except for the slash-character (/) directive), but enter DCL and CCL commands without a preceding character, unless it is the dollar sign character.

   b. Complete all commands on one line, whether they are DCL commands, CCL commands, or commands that run system utility programs. For commands that query or prompt, use either /NOQUERY or the equivalent utility option, or enter the responses to the prompts at the terminal. The following example shows an incorrect method of running a utility program from within a control file; the command is not complete on one line:

   ```
   RUN MACRO
   PROG=PROGA
   ^C
   ```

   c. Complete a command line with a RETURN. The last line of code in a control file must have a RETURN at its end if you want IND to process it.

2. **Labels**

   a. When defining labels, precede them with a period and end them with a colon.

   b. Labels must be placed at the beginning of a line, can be from 1 to 6 characters long, and can contain only alphanumeric or dollar sign ($) characters.

   c. When you reference labels in a command line (that is, you place them after a directive), use only the label without the period and colon.

3. **Comments**

   a. Place comments only on the end of a line or on a line by themselves.

   b. Never place IND comments on a line with DCL or CCL commands.

   c. Precede internal comments (those that are not displayed when the control file is run) with a period and a semicolon.

   d. Precede external comments (those you want displayed when the control file is run) with a semicolon.

4. **Symbols**

   a. Give symbols names from 1 to 6 characters in length.

   b. Include only alphanumeric characters and/or dollar signs ($) in symbols.

   c. To assign a character string to a symbol, enclose the string with quotes.

   d. To enable IND to substitute a value for a symbol, enclose the symbol with apostrophes and make sure your control file is in .ENABLE SUBSTITUTION mode (the default).

5. **Spacing**

   a. Separate directives from their arguments and from DCL or other commands by at least one space or tab, unless otherwise indicated in this manual.

   b. Use tabs and spaces to format control files for readability. You can insert spaces and horizontal tabs in a command line in the following locations:

      - At the start of the command line
      - Immediately following the colon (:) of a label
      - Directly before the semicolon (;) or period-semicolon (.;) of a comment

6. **Characters**

   a. IND accepts up to 132 characters in a command line.

   b. Include no more than 80 characters in .ASK, .ASKN, and .ASKS prompts. IND accepts up to 80 characters in these prompts.

Chapter 3

Executing a Control File

By default, IND.SAV must be on the system device to run IND. To change the device from which IND is run, see customization 39 in the *RT–11 Installation Guide*.

How to Execute a Control File

You can execute a control file from the keyboard monitor in any one of the following four ways:

- Call IND with the R command and then separately issue an IND command. For example:

  ```
  .R IND
  *
  ```

  The Command String Interpreter (CSI) displays an asterisk at the left margin of the terminal and waits for you to enter a command string. If you only press RETURN, IND displays its current version number and CSI prompts you again for a command string.

  The syntax of the command string is:

  ```
  control-filespec[/options]  [parameter1, parameter2, ...]
  ```

- Call IND with the RUN command and control file:

  ```
  .RUN IND control-filespec[/options]  [parameter1, parameter2, ...]
  ```

- Call IND with the control file:

  ```
  .IND control-filespec[/options]  [parameter1, parameter2, ...]
  ```

  This method and the next are the simplest ways to run IND.

- Issue the SET KMON IND command and then execute a control file with the @ key in response to the monitor dot prompt:

  ```
  @control-filespec[/options]  [parameter1, parameter2, ...]
  ```

  The default setting for the SET KMON command is SET KMON NOIND. This means that by default KMON interprets the *@filename* construction as DCL command-file syntax with a command file for KMON to execute.

  Once you issue the SET KMON IND command, the keyboard monitor interprets the *@filename* construction as control-file syntax and passes the specified file to IND.SAV for execution. To execute DCL command files under this condition, use the syntax, $@filename.

Restriction

You cannot run IND from VBGEXE. VBGEXE creates a copy of the fixed area, a pseudomonitor environment to make more memory available. When IND passes a command to the monitor, IND modifies locations in the fixed area of the monitor (not the VBG pseudomonitor environment). If you run IND from VBGEXE, when IND attempts to pass a command to the monitor, it will not find the fixed memory location it wants, since the VBG pseudomonitor environment is different, and your system will crash.

How IND Completes a Control File

When IND completes execution of a control file and any nested control files, it displays the message:

```
@ <EOF>
```

Then IND returns control to the keyboard monitor, which displays the period prompt.

Through a customization patch, you can eliminate or change the IND control-file terminating message. See customization 42 in the *RT–11 Installation Guide* for instructions on how to do this.

How to Abort a Control File

To abort IND from its asterisk (*) prompt, or whenever it is prompting you for input (from a control file), press CTRL/C once. To abort a control file while it is running, press CTRL/C twice.

You can disable the use of CTRL/C to abort IND, but, to do so, you must include global .SCCA support in your monitor through the system generation process. See the ABORT mode description under the .ENABLE/.DISABLE directive description for further information.

Testing the Execution of a Control File

You can execute IND directives on a line by line basis at the asterisk (*) prompt. To do so, run IND from the terminal by specifying the terminal device (TT:) as the control-file specification. IND then prompts you with the asterisk prompt for a single control line to execute:

```
.IND TT:
*
```

Since IND processes one line at a time when run from the terminal, labels have no meaning. Branch directives produce no results. This function is especially useful for testing the results of individual control lines.

In the following example, IND executes one control line and prompts for input:

```
.IND TT:
*.ASKS NAM WHAT IS YOUR NAME? RET
* WHAT IS YOUR NAME? [S] Jon RET
*
```

You can also use the trace (T) option of the IND control line to test the execution of control lines in an IND control file and to check for errors. See the *IND Command Options* section for an explanation of the trace option.

If you run IND without a file specification and without TT: in place of the file specification, IND prompts you with an asterisk prompt as in the preceding example. But, in response to the prompt, IND expects a control file to run. That is, IND then looks for a command string containing a control file specification. If you give it an IND directive instead, IND displays an error message.

The following sections describe the IND command string.

IND Command String

The format of the full IND command string is:

control-filespec[/options] [parameter1, parameter2, ...]

where:

control-filespec	The control file you want to execute. The default file type is COM. You can change this default through a software customization. See the *RT–11 Installation Guide* for instructions on how to do this.
options	One or more of the options listed in Table 3–1. The options you specify are stored in the local string symbol COMMAN with the entire command line, and in local string symbol P0. See the *Passing Parameters to a Control File* section for more information on COMMAN and P0.
parameters	One or more arguments (up to nine) that you want to pass to the control file. The parameters you specify are stored in local string symbols P1 through P9. Separate the parameters you specify with a space. See the *Passing Parameters to a Control File* section for more information on passing parameters.

IND Command Options

IND options allow you to change the way IND processes and displays a control file. Table 3–1 describes the options you can use in an IND command string.

Table 3–1: IND Options

Option	Name	Function
/D	Delete	Deletes the control file when IND has finished processing that file. The processing of a control file is complete only when IND executes the .EXIT directive or reaches the end of the control file. Therefore, if you use the /D option and IND encounters a .STOP directive or a slash (/) character, the file is not deleted. The .DISABLE DELETE directive in a control file overrides the /D option.
/N	Ignore	Directs IND to suppress execution of all keyboard commands in the control file. When you use the /N option, IND processes all the directives in a control file, but does not execute any of the keyboard commands. This function is useful if you wish to test the correct syntax and logical flow of your control file without executing any of the keyboard commands within. However, /N does not necessarily ensure correct logic for the whole control-file procedure. If, for example, you have a directive that branches as a result of the successful termination of a keyboard command, but that previous command was not executed, then the logic of the branch is not tested. If /N is used when calling a nested control file, keyboard commands are suppressed until the previous control file is reentered or an .ENABLE DCL or .ENABLE MCR directive is found in the nested control file.
/Q	Quiet (or) Suppress	Suppresses the display of keyboard commands and their results as IND executes the keyboard commands. The /Q option remains in effect even when control passes to a nested control file. A .DISABLE QUIET directive in the control file overrides the /Q option.

Table 3–1 (Cont.): IND Options

Option	Name	Function
/T	Trace	Displays each control line as it is processed. This tracing option is useful for testing and debugging control files. When you enable tracing, IND displays an exclamation mark (!) on the terminal before each line that begins with a directive. IND does not display any characters before comments and keyboard commands as they are processed. IND displays processed lines. Lines not processed as the result of a branch are not displayed. The /T option has the same effect as the .ENABLE TRACE directive in a control file and can be overridden from within the control file by the .DISABLE TRACE directive.

Passing Parameters to a Control File

When you execute a control file, you can pass it up to nine parameters by placing them after the control-file specification. Use a space to separate the parameters from each other and from the file specification.

IND assigns parameter values to the local IND string symbols P1 through P9. IND assigns a null value to any string symbols not used (if you have less than nine parameters). Since IND assigns values to the parameter symbols in the order in which IND receives those values, any symbols assigned a null value are always at the end of the parameter list of symbols. Note the following command line:

```
.IND ADINFO TODAY.DAT STORED.DAT
```

When the control file ADINFO.COM is executed, the parameters TODAY.DAT and STORED.DAT are passed. IND places the string TODAY.DAT in symbol P1 and the string STORED.DAT in symbol P2. This means the command procedure ADINFO can use the symbols P1 and P2 to respectively refer to the first and second file passed to ADINFO in the command line.

Note the contents of the file ADINFO.COM:

```
.; File name: ADINFO.COM
.; Purpose: This control file appends file1 to file2 and names the
.;          resulting file with the same name as file2.
.;
$COPY/CONCATENATE 'P1'+'P2' 'P2'
.EXIT
```

When .ENABLE SUBSTITUTION is in effect (which is the default), the apostrophe around each parameter symbol tells IND to substitute the value of the symbol for the symbol. This control file adds the contents of the first parameter (file1) to the second parameter (file2).

The advantage of having a control file to which you can pass parameters is that you can change the operation of the file without having to change the file.

Parameter symbols can be local or global. You will want to make one global if you intend to use it in a nested control file.

Prompting for Input

You may find it useful to write control files that can either accept parameters or prompt for user input if the required parameters are not specified.

For example, the following control file prompts for input if no parameter is specified. Since DCL prompts you for input with the COPY command, the purpose of this example is only to show you in a simple way how you can use parameters:

```
.; File name: MYCOPY.COM
.; Purpose:   Copies files and prompts
.;            for file names if names are
.;            not passed as parameters.
.;
.100:
.IF P1 = "" .ASKS P1 File to be copied?
.IF P1 = "" .GOTO 100  .;To ensure that parameter one be given.
.200:
.IF P2 = "" .ASKS P2 To where do you want 'P1' copied?
.IF P2 = "" .GOTO 200  .;To ensure that parameter two be given.
COPY 'P1' 'P2'
.EXIT
```

The following example shows how the preceding control file works:

```
.IND MYCOPY
* File to be copied? [S]: RET
* File to be copied? [S]: MEMO1.TXT RET
* To where do you want MEMO1.TXT copied?  DU1: RET
 Files copied:
 DK:MEMO1.TXT to DU1:MEMO1.TXT
 @ <EOF>
```

Storing Parameter Input According to Order Entered

IND stores information in parameter symbols in the order in which the information is entered. This means that if the line `.IF P1 = "" .GOTO 100` were not in the preceding code sample and the user entered only a RETURN at the first prompt, then P1 would contain a NULL value and the program MYCOPY would proceed to the next question: `To where do you want copied?`

The file name is missing from the question since none was given. If the user were then to respond to the second prompt with a file specification, IND would store the file specification in P2. But, when the program came to the DCL command line `COPY 'P1' 'P2'`, KMON would only see P2 and would therefore prompt: `To ?`

Specifying a Parameter As an Integer

Although IND always passes parameters as strings, you can specify a number as a parameter. Pass a numeric parameter if you want to pass a numbered list, a vector address, a CSR address, or a device number to a control file.

When you specify a numeric parameter, IND stores the parameter as a string value in one of the symbols P1 through P9. You can then use the string value of the numeric parameter or you can convert the value back to numeric. Use the string value if you want to display, print, or store the number. Use the numeric value if you want to perform some numeric function with it.

To use the parameter as a number, you need to convert it back to a numeric value. In the following example control file ADD.COM, the SETN and SETD directives do that conversion; SETN sets the value as numeric and SETD gives that value a decimal radix:

```
.;File Name: ADD.COM
.;Purpose:   To add small integer parameters; to show that parameters
.;           can be integers.
.;
.SETD A
.SETD B
.SETD C
.SETN A ’P1’
.SETN B ’P2’
.SETN C A+B
; ’A’ + ’B’ = ’C’
```

The following example shows a sample run of the file ADD.COM:

```
.IND ADD 24 25 RET
; 24 + 25 = 49
@ <EOF>
```

Executing a Control File from Within a Control File

You can call control files from within a control file. This process is called *nesting control files*; the control file that is called from the first control file is referred to as nested. You can nest up to three control files, for a total of four levels of control files (including the first file).

Use a command line with the following syntax to call a control file from within a control file:

@control-filespec[/options]   [parameter1, parameter2, ...]

where:

control-filespec	Specifies the control file to which you want to nest. The default file type is .COM.
options	Is one or more of the options listed in Table 3–1. The options you specify are stored in the local string symbol COMMAN with the entire command line, and in local string symbol P0. See the *IND Command-Line Symbols* section in Chapter 2 for more information on these symbols.
parameter[s]	Is one or more parameters (up to nine) that you want to pass to the control file. The surrounding brackets in the syntax illustration, which indicate that the parameters are optional, are not part of the command syntax. The parameters you specify are stored in local string symbols P0 through P9. See the *Passing Parameters to a Control File* section for more information on passing parameters.

You cannot include either internal or external comments on this command line.

The .ENABLE GLOBAL directive allows you to define symbols as global to all nested control-file levels. See the *Scope of IND Symbols* section in Chapter 2 for information on defining global symbols. If you do not use this directive, each time IND enters a deeper level it masks all symbols defined by the previous level out of the symbol table, so that only symbols defined in the current level are available. These symbols are recognized only within the level of the control file in which the symbols are defined. When control returns to a previous level, the symbols defined in that level become available again and the symbols from the lower levels are lost.

Executing a Command File from Within a Control File

The method for calling a command file from within a control file is similar to calling a control file from a control file (see the previous section). The one difference is that while you can pass parameters to a control file, you cannot pass parameters to a command file.

You can invoke a command file from a control file by placing a $@ sequence before the name of the command file you wish to invoke. When you pass control to a command file, the keyboard monitor processes and executes the file. When the command file has completed execution, control returns to the control file from which the command file was called. You cannot give arguments to the command file.

The command to invoke a command file is:

$@filespec

where:

filespec	Specifies the command file you wish to invoke. The default file type is COM.

For example, the following command line invokes the command file DUOUT.COM:

```
$@DUOUT
```

After the keyboard monitor has finished executing the command file, IND resumes processing of the control file from which the command file was called.

Control files cannot include a command file that calls another control file.

Executing Multiline Commands from a Control File

Control files cannot include CSI commands or DCL commands that require more than one line in control files. However, you can execute CSI commands and multiline DCL commands from a control file in two ways:

- By creating a command file that contains the command and calling the command file from the control file by using the $@ syntax.
- By creating a CCL command.

Creating a Command File

The following control file executes a CSI command by creating a command file, and then calling that file:

```
.OPEN SECOND.COM
.DATA R PIP
.DATA A.MAC=B.MAC
.DATA ^C
.CLOSE
$@SECOND
```

Creating a CCL Command

The following CCL command example achieves the same result as the previous example command file:

```
PIP A.MAC=B.MAC
```

Passing Commands Through a Chain Exit

If you pass more than one command through a special chain exit, you can call a command file only as the last command in that series of commands.

Running IND from a Mapped Monitor

When running under a mapped monitor, IND stores context information in a region of high memory, resulting in a performance improvement and saving you some low-memory space for other programs. For that reason, it is good to run IND under a mapped monitor.

IND determines if it is running under a mapped monitor; and if it is, a portion of IND is mapped to extended memory and resides there while IND is running. That portion is displayed by the SHOW MEMORY command issued while IND is running.

Through a software customization, you can suppress the dynamic allocation of a region of extended memory for IND. See customization 56 in the *RT–11 Installation Guide* for instructions on how to do this.

Removing IND from Extended Memory

When IND performs a normal exit, the portion of IND resident in extended memory is removed, and the area of extended memory previously taken by IND becomes available to other programs.

However, when you exit IND using a double CTRL/C, the portion of IND resident in extended memory remains in extended memory. Enter the following command in response to the monitor prompt (.) to remove IND from extended memory:

```
REMOVE IND
```

Part II
Descriptions of IND Directives

Part II alphabetically lists IND directive descriptions. However, the following related pairs of directives are described and alphabetically listed together under the first directive name of the pair, since their descriptions involve the same material and have similar examples:

.AND/.OR
.ENABLE/.DISABLE
.IFDF/.IFNDF
.IFENABLED/.IFDISABLED
.IFLOA/.IFNLOA
.SETD/.SETO
.SETT/.SETF

Chapter 4

IND Directives

This chapter first lists all the directives, briefly defining each, and then alphabetically describes each directive at length.

Directive Summaries

This section contains two IND directive summaries. Table 4–1 is an alphabetical summary; Table 4–2 is a type summary. Each table lists all the directives.

Table 4–1: Alphabetical Directive Summary

Directive	Function
/	Terminates control-file processing.
.label:	Assigns a name, represented by *label*, to a line in the control file so that the line can be referenced.
.AND	Combines logical tests on one command line so that the outcome of both tests joined by .AND determines the resulting action.
.ASK	Displays a prompt and uses the response to define or redefine a logical symbol and assign the symbol a logical (true or false) value.
.ASKN	Displays a prompt and uses the response to define or redefine a numeric symbol and assign it a numeric value.
.ASKS	Displays a prompt and uses the response to define or redefine a string symbol and assign it a string value.
.BEGIN	Marks the beginning of a begin-end block.
.CHAIN	Closes the current control file, opens another control file, and resumes execution.
.CLOSE	Closes an output data file.
.DATA	Specifies a single line of data to be sent to an output data file.
.DEC	Subtracts one from the value of a numeric symbol.
.DELAY	Delays control-file processing for a specified period of time.
.DISABLE	Disables the operating modes. See the description of the .ENABLE directive for descriptions of operating modes.
.DUMP	Displays local, global, and special symbol definitions.

Table 4–1 (Cont.): Alphabetical Directive Summary

Directive	Function
.ENABLE	Enables the operating modes. See the description of the .ENABLE directive for descriptions of operating modes.
.END	Marks the end of a begin-end block.
.ERASE	Deletes local or global symbols from the symbol tables.
.EXIT	Terminates processing of either the current control file or a begin-end block, and returns control to the previous level; can also assign a value to the numeric symbol <EXSTAT> (see the <EXSTAT> special symbol description in Table 2–2).
.GOSUB	Branches to a subroutine within the control file.
.GOTO	Branches to another location in the control file.
.IF	Determines if a symbol satisfies one of several possible conditions.
.IFDF .IFNDF	Determines whether a symbol is defined or not defined.
.IFENABLED .IFDISABLED	Determines whether an operating mode is enabled or disabled. See the description of the .ENABLE directive for descriptions of operating modes.
.IFLOA .IFNLOA	Determines whether a device handler has been loaded or not loaded.
.IFT .IFF	Determines whether a logical symbol is true or false, or tests specific bits in a numeric symbol.
.INC	Adds one to the value of a numeric symbol.
.ONERR	On detecting an error, branches to another location in the control file.
.OPEN	Creates an output data file. If the file you specify with .OPEN already exists, .OPEN creates a new file and will delete the existing file if you subsequently use the .CLOSE directive. Use the .OPEN directive only when you write to a file.
.OPENA	Opens an existing file and appends subsequent data to it. If the file you specify does not exist, .OPENA creates a new file. Use this directive only when you write to a file.
.OPENR	Opens an existing file for use with the .READ directive. Use this directive only when you read from a file.
.OR	Combines logical tests on one command line so that the outcome of either or both tests determines the resulting action.
.PARSE	Breaks a string into substrings.
.PURGE	Discards or closes an output file without making any changes to the file.
.READ	Reads the next record from a file into a string variable. The file must have been previously opened with .OPENR.

Table 4–1 (Cont.): Alphabetical Directive Summary

Directive	Function
.RETURN	Returns control from a subroutine to the line immediately following that subroutine's call.
.SETD	Redefines a numeric symbol to a decimal radix.
.SETL	Defines or redefines a logical symbol and assigns it a logical value.
.SETN	Defines or redefines a numeric symbol and assigns it a numeric value.
.SETO	Redefines a numeric symbol to an octal radix.
.SETS	Defines or redefines a string symbol and assigns it a string value.
.SETT .SETF	Defines or redefines a logical symbol or redefines bits within a numeric symbol, and assigns the symbol or bits a true or false value.
.STOP	Terminates control-file processing.
.STRUCTURE	Determines the file structure of a specified file-structured device.
.TEST	Tests attributes of a symbol or string and stores the results in special symbols.
.TESTDEVICE	Tests a specified device and stores the device attributes in the special symbol <EXSTRI>.
.TESTFILE	Determines if a file exists and stores the results in the special symbols <FILSPC> and <FILERR>.
.VOL	Assigns a volume ID to a string symbol.

Table 4–2: Directive Summary by Type

Directive	Function
Label-Definition	
.label:	Assigns a name, represented by label, to a line in the control file so that the line can be referenced.
Symbol-Definition	
.ASK	Displays a prompt and uses the response to define or redefine a logical symbol and assign the symbol a logical (true or false) value.
.ASKN	Displays a prompt and uses the response to define or redefine a numeric symbol and assign it a numeric value.
.ASKS	Displays a prompt and uses the response to define or redefine a string symbol and assign it a string value.
.DUMP	Displays local, global, and special symbol definitions.

Table 4–2 (Cont.): Directive Summary by Type

Directive	Function
.ERASE	Deletes local or global symbols from the symbol tables.
.PARSE	Breaks a string into substrings.
.SETD .SETO	Redefines a numeric symbol to a decimal (.SETD) or an octal (.SETO) radix.
.SETL	Defines or redefines a logical symbol and assigns it a logical value.
.SETN	Defines or redefines a numeric symbol and assigns it a numeric value.
.SETS	Defines or redefines a string symbol and assigns it a string value.
.SETT .SETF	Defines or redefines a logical symbol or redefines bits within a numeric symbol, and assigns the symbol or bits a true or false value.
.STRUCTURE	Determines the file structure of a specified file-structured device.
.TEST	Tests attributes of a symbol or string and stores the results in special symbols.
.TESTDEVICE	Tests a specified device and stores the device attributes in the special symbol <EXSTRI>.
.TESTFILE	Determines if a file exists and stores the results in the special symbols <FILSPC> and <FILERR>.
.VOL	Assigns a volume ID to a string symbol.
File-Access	
.CHAIN	Closes the current control file, opens another control file, and resumes execution.
.CLOSE	Closes an output data file.
.DATA	Specifies a single line of data to be sent to an output data file.
.OPEN	Creates an output data file. If the file you specify with .OPEN already exists, .OPEN creates a new file and will delete the existing file if you subsequently use the .CLOSE directive. Use the .OPEN directive only when you write to a file.
.OPENA	Opens an existing file and appends subsequent data to it. If the file you specify does not exist, .OPENA creates a new file. Use this directive only when you write to a file.
.OPENR	Opens an existing file for use with the .READ directive. Use this directive only when you read from a file.

Table 4–2 (Cont.): Directive Summary by Type

Directive	Function
.PURGE	Discards or closes an output file without making any changes to the file.
.READ	Reads the next record from a file into a string variable. The file must have been previously opened with .OPENR.
Logical-Control	
.BEGIN	Marks the beginning of a begin-end block.
.END	Marks the end of a begin-end block.
.EXIT	Terminates processing of either the current control file or a begin-end block, and returns control to the previous level; can also assign a value to the numeric symbol <EXSTAT> (see the <EXSTAT> special symbol description in Table 2–2).
.GOSUB	Branches to a subroutine within the control file.
.GOTO	Branches to another location in the control file.
.ONERR	On detecting an error, branches to another location in the control file.
.RETURN	Returns control from a subroutine to the line immediately following that subroutine's call.
.STOP or /	Terminates file processing.
Logical-Test	
.AND .OR	Combines logical tests on one line. .AND combines logical tests so that the outcome of both tests determines the resulting action. .OR combines logical tests so that the outcome of either or both tests determines the resulting action.
.IF	Determines whether a symbol satisfies one of several possible conditions.
.IFDF .IFNDF	Determines whether a symbol is defined or not defined.
.IFENABLED .IFDISABLED	Determines whether an operating mode is enabled or disabled. See the description of the .ENABLE directive for descriptions of operating modes.
.IFLOA .IFNLOA	Determines whether a device handler has been loaded or not loaded.
.IFT .IFF	Determines whether a logical symbol is true or false, or tests specific bits in a numeric symbol.

Table 4–2 (Cont.): Directive Summary by Type

Directive	Function
Execution-Control	
.DELAY	Delays control-file processing for a specified period of time.
Operating-Mode	
.ENABLE .DISABLE	Enables or disables the operating modes. See the description of the .ENABLE directive for descriptions of operating modes.
Decrement/Increment	
.DEC	Subtracts one from the value of a numeric symbol.
.INC	Adds one to the value of a numeric symbol.

/ (end-of-file directive)

The logical end-of-file directive, the slash character (/), terminates file processing.

Description

The slash character performs the same function as the .STOP directive. You can use this directive at any location in the control file. IND ignores any characters that follow the slash on the same line.

When IND encounters the slash, IND displays the following message at the console:

```
@ <EOF>
```

You cannot assign an exit status value when you use the slash character.

Example

The following example uses the end-of-file directive:

```
       .ASK CONT DO YOU WISH TO CONTINUE
       .IFT CONT .GOTO 100
       /
.100:
```

In this example, execution halts if the logical symbol CONT is defined as false.

.label: (label directive)

The label directive, a one- to six-character name of your choice beginning with a period and ending with a colon, assigns a name to a line in your control file so that the line can be referenced.

Format

.label:

Description

A label consists of only alphanumeric characters and/or a leading, trailing, or embedded dollar sign ($). Labels must:

- Be from one to six characters prefixed with a period (.) and followed by a colon (:).
- Always appear at the beginning of a line, and must be the only label on a line.

Nested Control Files

A label is known only within the level of the control file in which that label is defined; that is, if you nest a control file within a control file, the labels within each control file are recognized only in their separate control files.

Two Types of Labels

When your control file instructs IND to branch to a label, IND first determines whether or not the label is a direct-access or a nondirect-access label. For each type of label, IND uses a different search method to find the line referenced by the label. If IND cannot find the label, it displays an error message.

The following are the two types of labels and the two methods IND uses to find the lines referenced by them:

- Direct-Access Labels

  A direct-access label is on a line by itself. IND can branch to direct-access labels more quickly than to nondirect-access labels, because IND records direct-access labels and their locations in an internal table. When a direct-access label is referenced, IND checks the direct-access table and jumps directly to the proper location without having to search the file. IND then continues processing at the statement directly below the direct-access label.

  You can define up to 20 direct-access labels within a control file. If you define more than 20, the newly defined labels replace the already defined labels in order, beginning with the first direct-access label defined. The replaced direct-access labels are then treated as nondirect-access labels.

- Nondirect-Access Labels

  A nondirect-access label is placed at the beginning of a line that also has an IND directive, a DCL command, or a comment.

  IND searches for a nondirect-access label from the current position in the file to the end of the file. If the label is not found, IND searches from the beginning of the file toward the current position. When IND finds the label, processing continues on that line.

  If IND does not find a label, IND reports an error and terminates the procedure.

Examples

1. In the following line, START is a label:

   ```
   .START: .ASKS ANS DO YOU HAVE A PRINTER?
   ```

2. In the following example, 100 is a direct-access label, while 200 is not:

   ```
   .100:
          .;THIS IS THE START OF A SUBROUTINE
                            .
                            .
                            .
                 .RETURN
   .200:         .ASK A DO YOU WANT TO CONTINUE
                 .IFT A .GOSUB 100
                            .
                            .
                            .
   ```

.AND/.OR

.AND combines logical tests in one command line so that the outcome of *both* tests joined by .AND determines whether or not to perform the action at the end of the command line.

.OR combines logical tests in one command line so that the outcome of *either* or *both* tests joined by .OR determines whether or not to perform the action at the end of the command line.

You can combine .OR directives with .AND directives on the same line, but IND processes .AND directives before .OR directives.

Format

.AND

.OR

Examples

1. Control passes to HELP if logical symbol A is true and logical symbol B is false:

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

2. You can omit the .AND directive; IND assumes an .AND directive between logical tests when you put more than one logical test on the same line. This example has the same effect as the previous one. IND assumes an .AND directive between the two tests .IFT A and .IFF B:

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

3. In the following example, control branches to the label HELP if A is true or if B is false:

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

4. IND processes .AND directives before .OR directives. When IND processes the first line in the following example, the effect is the same as the second line:

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

.ASK

Sets a TRUE or FALSE value of a *logical* symbol based on a user response.

Format

.ASK *[default-response:time] logical-symbol prompt*

where:

default-response

Specifies the default response that is used when only a RET is entered as a response, or when a specified time interval elapses and no response has been given.

Specify the default response by using a logical or special symbol (such as <TRUE> or <FALSE>) that is assigned a true or false value.

If you do not specify a default response, the default response is NO.

time

Specifies the timeout count. If the timeout count is exceeded and no response is given, IND uses the default response and the special symbols <DEFAUL> and <TIMOUT> are set to true.

Note

You can use a timeout count only if:

- You have enabled timeout mode by means of the .ENABLE TIMEOUT directive.
- Your configuration includes a system clock and your monitor includes timer support.

If you specify a timeout count in a control file, and either of these two conditions is not met, the timeout indicator is not displayed in the resulting prompt and the timeout count is ignored.

The timeout-count syntax is:

nnu

where:

nn	Specifies the number of time units to count before the timeout occurs. IND interprets *nn* as an octal number unless you use a decimal point (.) following the number to denote decimal.

u Specifies one of the following time units:

T	Ticks
S	Seconds
M	Minutes
H	Hours

If you specify an invalid timeout count, an error occurs.

logical-symbol

Specifies a logical symbol to be set to true or false.

prompt

Specifies the question to be displayed at the console. The prompt you specify can include up to 80 characters.

Description

The .ASK directive:

- Displays a question at the terminal.
- Waits for a YES or NO response.
- Sets a specified *logical* symbol to a value of TRUE for yes or FALSE for no.

  If the symbol has not already been defined, IND makes an entry in the symbol table. If the symbol has been defined, IND resets its value in the symbol table according to the response. IND displays an error message and exits if the symbol was previously defined as a numeric or string symbol.

Note

- The brackets surrounding the optional parameters *default-response* and *time* are part of the syntax. You must include them if you specify a value for either parameter.
- Although both of the preceding parameters are optional, they are position dependent within the brackets. If you specify time without specifying a default, you must delimit the position of the default parameter with a colon (:).

How IND Prompts with a YES/NO Question

When IND processes an .ASK directive in a command line, IND prompts the user of the terminal with a question. The prompt displayed by IND contains the following information:

An asterisk (*)
The question you specified (prompt)
A question mark (?)
Response information in brackets (taken from the optional parameters)

For example, when IND processes the following command line:

```
.ASK [:15.S] DONE ARE YOU FINISHED
```

IND prompts:

```
* ARE YOU FINISHED? [Y/N D:N T:15.S]
```

- The Y/N indicates that a YES or NO response is required.
- The notation D:N indicates that the default response is NO.
- The notation T:15.S indicates that the default response will be used if no response is given within 15_{10} seconds.

How to Respond to a YES/NO (.ASK) Question

Response	IND Reaction
Y	If you respond with any string beginning with a Y, IND interprets it to mean YES, and sets the specified logical symbol to a value of TRUE.
N	If you respond with any string beginning with an N, IND interprets it to mean NO, and the logical symbol is set to a value of FALSE. A response beginning with any character other than N or Y causes IND to redisplay the question.
RET	If you respond with only RET, IND uses the default response indicated within the brackets. This response sets <DEFAUL> to TRUE.
ESC RET	If you respond with ESC RET while escape recognition is enabled (with the .ENABLE ESCAPE command), the special symbol <ESCAPE> is set to TRUE if it has not been previously defined. If the escape symbol has been previously defined, its value remains unchanged. If you respond with ESC RET while escape recognition is disabled (by the .DISABLE ESCAPE directive), <ESCAPE> is set to FALSE and IND displays an error message.
CTRL/Z	If you respond with CTRL/Z, IND displays the following message and terminates processing: `@EOF`

Example

The following question directive will prompt: ARE YOU FINISHED. It will expect a Y (yes) or N (no) for an answer. IND will place the answer in the logical symbol DONE. Since no default response is specified, the default response for this question is NO. If the user does not respond in 15 seconds, IND will use the default response (NO); that is, IND will assign the value false (for no) to the logical symbol DONE.

This example presumes you have enabled timeout and you have a system clock with timer support:

```
.ASK [:15.S] DONE ARE YOU FINISHED
```

.ASKN

Sets a numeric value of a specified *numeric* symbol based on a user response.

Format

.ASKN *[low:high:default-response:time]* *numeric-symbol* *prompt*

where:

low:high

Specifies the inclusive numerical range within which the response must fall. The default range is 0 through 177777_8, or 0 through 65535_{10}. If you specify values for low and high, they must fall within the default range. You can specify these values as numbers or as numeric expressions.

default-response

Specifies the default response that is used when only a RET is entered as a response, or when a specified time interval elapses and no response has been given. You can specify the default response either as a number or as a numeric expression. If no default response is specified, the default response is assigned the value of the low limit of the range (either the assigned range or the default range if none is assigned).

time

Specifies the timeout count. If the timeout count is exceeded and no response is given, IND uses the default response and the special symbols <DEFAUL> and <TIMOUT> are set to TRUE.

Note

You can use a timeout count only if:

- You have enabled timeout mode by means of the .ENABLE TIMEOUT directive.
- Your configuration includes a system clock and your monitor includes timer support.

If you specify a timeout count in a control file, and either of these two conditions is not met, the timeout indicator is not displayed in the resulting prompt and the timeout count is ignored.

The timeout-count syntax is:

nnu

where:

nn	Specifies the number of time units to count before the timeout occurs. IND interprets *nn* as an octal number unless you use a decimal point (.) following the number to denote decimal.
u	Specifies one of the following time units:

```
T  Ticks
S  Seconds
M  Minutes
H  Hours
```

numeric-symbol

Specifies a numeric symbol to be assigned the value of the response.

prompt

Specifies a string of characters to be displayed at the console. If the prompt is a question, you must include the question mark as part of the prompt. The prompt you specify can include up to 80 characters.

Description

The .ASKN directive:

- Displays a question or prompt on the terminal.
- Waits for a numeric response.
- Sets the value of the specified numeric symbol to the value of the response.

  If the symbol has not already been defined, IND makes an entry in the symbol table. If the symbol has been defined, IND resets its value in the symbol table according to the response and the default radix mode enabled (octal or decimal). IND displays an error message and exits if the symbol was previously defined as a logical or string symbol.

Note

- The brackets surrounding the optional parameters *low*, *high*, *default-response*, and *time* are part of the syntax; you must include them if you specify a value for any of these parameters.
- Although all the bracketed parameters are optional, they are position dependent within the brackets. You must use a colon to delimit the position of any parameter you exclude if you want to specify a parameter that follows it within the brackets.

How to Set the Radix (octal or decimal) of the Range and Timeout Values

The .ENABLE/.DISABLE OCTAL directive sets the radix for the numbers you use with IND directives. This radix determines how the range, the default, and the timeout indicators are displayed in a resulting prompt, and how you should respond (with octal or decimal numbers).

If no specific radix has been ENABLEd, the default radix is octal, and IND considers the indicators you specify to be octal. However, if a decimal radix has been ENABLEd (by .DISABLEing OCTAL), IND interprets the responses you supply as decimal.

You can override an octal radix by placing a decimal point (.) after any of the values you specify within the brackets. IND considers values that you specify with a decimal point (.) to be decimal values. Any values within the same set of brackets specified without a decimal point are interpreted as octal, but IND converts them to their decimal equivalents before displaying the resulting prompt.

When a decimal radix is in effect, all values specified within the brackets are considered decimal; using a decimal point (.) has no effect.

When you use numeric symbols or expressions to specify the range or default response, the radix of the numeric symbols determines the radix of the range and default values.

How IND Displays a Numeric (.ASKN) Question

When IND processes an .ASKN directive in a command line, IND displays:

An asterisk (*) followed by a space—unless .DISABLE PREFIX has been specified
The prompt you specified
The response indicators in brackets, taken from the optional values you specified—unless .DISABLE SUFFIX has been specified

For example, when IND processes the following command line:

```
.ENABLE OCTAL
.ASKN [0:10:3.:20.S] ERR NO. OF ERROR CODES TO USE
```

IND displays:

```
* NO. OF ERROR CODES TO USE [D R:0.-8. D:3. T:20.S]
```

Since decimal values were specified within the brackets in the original command line, all values within the resulting prompt are shown as decimal. The decimal points (.) following the values for the range, default response, and timeout count indicate that these are decimal numbers.

- The D indicates that the default radix for the response is *decimal*, and that the response will always be stored as a decimal number.

  If octal values had been specified in the original command line, IND would display the O (octal) indicator instead, meaning the default radix of the response is octal and the response will be stored as an octal value.

- The R: specifies the *range* of characters allowed in the response string. The notation R:0.-8. indicates that the value must be a number in the range *0 to* 8_{10} *inclusive*.
- The D: specifies the *default* response string. The notation D:3. indicates that the default response is *3*.
- The T: specifies the given response *time*. The notation T:20.S indicates that the default response will be used if no response is given within 20_{10} *seconds*.

How to Respond to a Numeric (.ASKN) Question

The response to an .ASKN question must be an octal or decimal number within the range specified by the prompt. The O or D radix indicator tells you the radix in which the response is stored.

Response	IND Reaction
Octal value	If the radix indicator within the prompt brackets is O, IND assumes the response is an octal value. You can specify a decimal value by typing a decimal point (.) after the response. IND stores the response as its octal equivalent.
Decimal value	If the radix indicator is D, IND assumes the response is a decimal value. You can specify an octal value by typing a number sign (#) before your response. IND stores the response as its decimal equivalent.
RET	If you respond with only RET, IND uses the default response indicated within the brackets. This response sets <DEFAUL> to TRUE.
ESC RET	If you respond with ESC RET while escape recognition is enabled (with the .ENABLE ESCAPE command), the special symbol <ESCAPE> is set to TRUE and the numeric symbol is set to 0 if the symbol has not yet been defined. If the escape symbol was previously defined, its value remains unchanged. If you respond with ESC RET while escape recognition is disabled (by the .DISABLE ESCAPE directive), <ESCAPE> is set to FALSE and IND displays an error message.
CTRL/Z	If you respond with CTRL/Z, IND displays the following message and terminates processing: `@EOF`

Examples

1. In the following command line:

   - :7 specifies a high value for the range but no low limit.
   - ::15.S specifies a 15-second timeout count; the extra colon means that no default response is specified.

   In this case, if no response is given within 15 seconds, IND assigns the value 0 (the default value of the low limit) to the numeric symbol NUM:

   ```
   .ASKN [:7::15.S] NUM # OF LINEPRINTERS IN CONFIGURATION?
   ```

2. In the following control file, an octal radix is enabled. However, the octal radix is overridden in the .ASKN directive since two of the numbers specified in that directive are specified as decimal. Furthermore, whenever one or more numbers within the brackets are specified as decimal, IND translates any other numbers within the same brackets into decimal. So, in processing the following example file, IND translates the octal 10 into the decimal 8.

   In the following file, the valid range for the response is 0-8_{10}, the default response is 3_{10}, and the timeout count is 20_{10} seconds:

   ```
   .ENABLE OCTAL
   .ASKN [0:10:3.:20.S] ERR NO. OF ERROR CODES TO USE
   ```

.ASKS

Sets an ASCII string value to a specified *string* symbol based on a user response.

Format

.ASKS *[low:high:"default-response":time]* *string-symbol* *prompt*

where:

low:high

Specifies the inclusive number of characters permitted in the response string. The default range is 0 through 204_8 or 0 through 132_{10}.

If you specify values for low and high, they must fall within the default range and they must be numbers or numeric expressions.

"default-response"

Specifies the default response that is used when only a RET is entered as a response, or when a specified time interval elapses and no response is given.

You can specify the default response either as a string, a string symbol, or a string expression. Use quotation marks only if you specify a string.

time

Specifies the timeout count. If the timeout count is exceeded and no response is given, IND uses the default response, and the special symbols <DEFAUL> and <TIMOUT> are set to TRUE.

Note

You can use a timeout count only if:

- You have enabled timeout mode by means of the .ENABLE TIMEOUT directive.
- Your configuration includes a system clock and your monitor includes timer support.

If you specify a timeout count in a control file, and either of these two conditions is not met, the timeout indicator is not displayed in the resulting prompt and the timeout count is ignored.

The timeout-count syntax is:

nnu

where:

nn Specifies the number of time units to count before the timeout occurs. IND interprets *nn* as an octal number unless you use a decimal point (.) following the number to denote decimal.

u Specifies one of the following time units:

T	Ticks
S	Seconds
M	Minutes
H	Hours

string-symbol

Specifies a string symbol to be assigned the value of the response.

prompt

Specifies a string of characters to be displayed at the console. If the prompt is a question, you must include the question mark as part of prompt. The prompt you specify can include up to 80 characters.

Description

The .ASKS directive:

- Displays a question or prompt at the terminal.
- Waits for an ASCII string response.
- Sets the specified string symbol to the value of the response.

  If the symbol has not already been defined, IND makes an entry in the symbol table. If the symbol has been defined, IND resets its value in the symbol table according to the response. IND displays an error message and exits if the symbol was previously defined as a logical or numeric symbol.

Note

- The brackets surrounding the optional parameters *low*, *high*, *default-response*, and *time* are part of the syntax; you must include them if you specify a value for any of these parameters.
- Although all the bracketed parameters are optional, they are position dependent within the brackets. You must use a colon to delimit the position of any parameter you exclude if you want to specify a parameter that follows it within the brackets. For example, in the following command line the *high* value is omitted, but its colon position indicator is kept:

```
.ASKS [3::"DU1":15.S] DEV DEVICE TO USE FOR DEFAULT?
```

How to Set the Radix (octal or decimal) of the Range and Timeout Values

The .ENABLE/.DISABLE OCTAL directive sets the radix for the numbers you use with IND directives. This radix determines how the range, the default, and the timeout indicators are displayed in a resulting prompt, and how you should respond (with octal or decimal numbers).

If no specific radix has been ENABLEd, the default radix is octal, and IND considers the indicators you specify to be octal. However, if a decimal radix has been ENABLEd (by .DISABLEing OCTAL), IND interprets the responses you supply as decimal.

You can override an octal radix by placing a decimal point (.) after any of the values you specify within the brackets. IND considers values that you specify with a decimal point (.) to be decimal values. Any values within the same set of brackets specified without a decimal point are interpreted as octal, but IND converts them to their decimal equivalents before displaying the resulting prompt.

When a decimal radix is in effect, all values specified within the brackets are considered decimal; using a decimal point (.) has no effect.

When you use numeric symbols or expressions to specify the range or default response, the radix of the numeric symbols determines the radix of the range and default values.

For example, in the following control file, octal mode is enabled. When IND processes the .ASKS directive, IND interprets the timeout value as a decimal number. Then to make the numbers in the IND prompt be consistent (have one radix for all of them), IND converts the given range values (0 to 10) from octal to decimal (0 to 8) when it prompts the user:

```
.ENABLE OCTAL
.ASKS [0:10:"RT11A":20.S] VOLID TYPE YOUR VOLUME ID
```

How IND Displays a String (.ASKS) Question

When IND processes an .ASKS directive in a command line, IND prompts the user of the terminal with a question. The prompt contains the following information:

An asterisk (*) followed by a space—unless .DISABLE PREFIX has been specified
The question you specified (prompt)
A question mark (?)
Response information in brackets, taken from the optional parameters—unless .DISABLE SUFFIX has been specified

For example, when IND processes the following command line:

```
.ENABLE OCTAL
.ASKS [0:10:"RT11A":20.S] VOLID TYPE YOUR VOLUME ID
```

IND displays:

```
* TYPE YOUR VOLUME ID [S R:0.-8. D:"RT11A" T:20.S]
```

- The S indicates that the response must be an ASCII *string*.
- The R: specifies the *range* of characters allowed in the response string. The notation R:0.-8. indicates that the response string can be from *0 to* 8_{10} *characters long*, inclusive.
- The D: specifies the *default* response string. The notation D:"RT11A" indicates that the default volume ID is *"RT11A"*. (If no default response was specified, the D: is not displayed.)

- The T: specifes the given response *time*. The notation T:20.S indicates that the default response will automatically be used if no response is given within *20*$_{10}$ *seconds*. Note that IND indicates decimal values by displaying a decimal point (.) and indicates octal values by excluding the decimal point.

How to Respond to a String (.ASKS) Question

Response	IND Reaction
ASCII string	If you respond with an ASCII string, its length should be within the range specified by the prompt.
RET	If you respond with only RET, IND uses the default response indicated within the brackets. This response sets <DEFAUL> to TRUE. If no default response has been specified, the symbol is set to null.
ESC RET	If you respond with ESC RET while escape recognition is enabled (with the .ENABLE ESCAPE command), the special symbol <ESCAPE> is set to TRUE and the string symbol is defined as null (if it has not been previously defined). If the string symbol has been previously defined, its definition remains unchanged. If you respond with ESC RET while escape recognition is disabled (by the .DISABLE ESCAPE directive), <ESCAPE> is set to FALSE and IND displays an error message.
CTRL/Z	If you respond with CTRL/Z, IND displays the following message and terminates processing: `@EOF`

Examples

1. The following command line specifies a low value of 3 for the range and omits a high value, defines DU1 as the default response, and gives a 15-second timeout count:

   ```
   .ASKS [3::"DU1":15.S] DEV DEVICE TO USE FOR DEFAULT?
   ```

   Since no high limit is specified, the default high limit of 204$_8$ characters will be accepted for an input string symbol. If no response is given within 15 seconds, IND assigns the value DU1 to the string symbol DEV.

2. In this control file, octal mode is enabled. However, since the time is expressed as a decimal number, IND translates the 10 octal into an 8 decimal when it prompts the user:

   ```
   .ENABLE OCTAL
   .ASKS [0:10:"RT11A":20.S] VOLID TYPE YOUR VOLUME ID
   ```

   In this case, the response must contain from 0$_{10}$ to 8$_{10}$ ASCII characters, the default response is RT11A, and the timeout count is 20$_{10}$ seconds.

.BEGIN

Marks the beginning of a begin-end block of code in an IND control file. See also the .END directive.

Format

.BEGIN

Description

Four features of the .BEGIN directive are:

- This directive has no arguments. Anything that follows a .BEGIN directive on the same line is ignored.
- Each .BEGIN directive must have a corresponding .END directive. That is, the block of code that a .BEGIN directive starts must be terminated by an .END directive.
- The .BEGIN and .END directives permit you to structure the control file in blocks. Modular, block-structured control files are easier to debug and maintain.
- Begin-end blocks isolate local symbol definitions and thus conserve symbol-table space. This is helpful if you have a large IND file with many symbols.

Defining Local and Global Symbols

When you define a symbol, IND creates an entry in an internal symbol table.

When you define a symbol within a begin-end block, that symbol is called a *local* symbol. Local symbols are erased from the symbol table when IND encounters an .END directive; that is, they exist only within the begin-end block in which they are defined.

When you define a symbol outside a begin-end block, that symbol is called a *global* symbol. Global symbols remain within the symbol table throughout the execution of a control file. Global symbols also remain within the symbol table throughout the execution of all control files nested within the control file in which they are defined.

Redefining Symbols

If a symbol is defined as a logical, numeric, or string symbol outside a begin-end block, you can redefine meaning and type of the symbol within the begin-end block. However, when you exit from the begin-end block, the redefined symbol is erased and the symbol returns to its previous meaning and type.

Erasing Local Symbols Within the Local Block

The .ERASE LOCAL directive erases all local symbols within the block containing that directive. See the description of that directive for more details.

Nesting Begin-End Blocks

Begin-end blocks can be nested up to a maximum depth of 127, but IND usually exhausts stack space before this limit can be reached.

Examples

1. The following control file can help you delete unnecessary files from your directory. With this file, IND asks for the name of the file you want to delete. However, before the file is deleted, the file name you entered is displayed on the screen and IND asks you if that is the file you wanted to delete. This verification ensures that you do not delete a file you really want to keep:

```
.BEGIN:
         .ASKS FILE Which File?
         $TYPE 'FILE'
         .ASK DEL Delete this file?
         .IFT DEL DELETE 'FILE'
         .GOTO BEGIN
.END
```

2. Example two shows how .BEGIN/END blocks casue local variables created inside them to have limited scope and existence. Note that the variable TRUTH, redefined inside the block, reverts to its original value when the block ends.

```
.SETT TRUTH
.IFT TRUTH        ; The truth shall make you free.
.BEGIN
.SETF TRUTH
.IFF TRUTH        ; Truth is falsehood, temporarily.
.END
.IFT TRUTH        ; Truth has triumphed.
```

   The preceding code returns the following three lines:

```
; The truth shall make you free.
; Truth is falsehood, temporarily.
; Truth has triumphed.
```

   This example shows the redefinition of TRUTH within the block and the automatic restoring of TRUTH's value when it exits the block.

.CHAIN

Passes the command processing to another control file and performs the following:

- Closes the current control file.
- Disregards all current local symbols.
- Continues processing by passing control to another control file.

Format

.CHAIN *filespec [/options]*

where:

filespec
Specifies the control file to which control is to be passed.

/options
Specifies one of the options you can use in an IND command string. These options are described in Table 3–1.

Description

The .CHAIN directive passes the command processing to another file. However, this directive does not close data files or change any nested control-file level.

Example

In the following example, IND passes control to the file DK:OUTPUT.COM:

```
.CHAIN OUTPUT
```

.CLOSE

Closes a file opened by the .OPEN, .OPENA, or .OPENR directive.

Format

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

where:

#n

Specifies an optional file number from 0 to 3; the default is 0.

The pound sign (#) preceding the *n* is part of the necessary syntax. Since a file name can begin with a numeric character, you need the # to tell IND that the following character is a file *number* rather than a file *name*.

The *n* is the file number. If substitution mode is enabled, you can substitute a symbol for *n* by enclosing the symbol in apostrophes.

filespec

Specifies the name of the file you are closing. If you opened the file specifying a non-zero file number, you must specify the file number in .CLOSE. The file name can be included for readability.

Description

Three features of the .CLOSE directive are:

- You must close any open files before passing control from IND to the keyboard monitor.
- The .CLOSE directive disables DATA mode.
- If you use the .CLOSE directive after .OPENR, .CLOSE has the same effect as the .PURGE directive.

Example

In the following example, the .CLOSE directive closes the file RMSG.TXT after the user reads it:

```
     .;Create file to be TYPEd
     .;as error message.
     .;Note that .OPEN cannot have
     .;a comment.
 .OPEN RMSG.TXT
 .DATA FILE OPEN READ-ONLY
 .CLOSE RMSG.TXT
.;
     .;Create data file with
     .;two lines of text.
 .OPEN #0 FILE.TXT
 .DATA #0 ABC
 .DATA #0 DEF
 .CLOSE #0
.;
     .;Invoke KMON to TYPE data file.
 $TYPE FILE.TXT
.;
 .ONERR 100   .;Specify where to go if error occurs.
.;
     .;Try to write to data file opened for
     .;read only.  Attempt will fail.
     .;Note that .OPENR cannot have
     .;a comment.
 .OPENR #1 FILE.TXT
 .DATA #1 GHI
 .GOTO 200

.100: .CLOSE #1   .;Error Handling: Must close open
     .;file before TYPEing error messsage.
 $TYPE RMSG.TXT
 .GOTO 300   .;Resume control file execution.

.200: .CLOSE #1   .;The GOTO 200 never done because
.;     .;of write error above.
.;
     .;Re-open data file and append
     .;two more lines.  No error since
     .;.OPENA allows write access.
     .;Note that .OPENA cannot have
     .;a comment.
.300: .OPENA #2 FILE.TXT
 .DATA #2 GHI
 .CLOSE #2   .;Close updated data file.
.;
     .;Invoke KMON to TYPE data file.
 $TYPE FILE.TXT
;
;       !!!!! ALL DONE !!!!!
 .STOP
```

.DATA

Writes a record to a file previously opened by an .OPEN or .OPENA directive.

Format

DATA *[#n]* *text-string*

where:

#n

Specifies an optional file number from 0 to 3; the default is 0.

The pound sign (#) preceding the *n* is part of the necessary syntax; since a file name can begin with a numeric character, you need the # to tell IND that the following character is a file *number* rather than a file *name*.

The *n* is the file number. If substitution mode is enabled, you can substitute a symbol for *n* by enclosing the symbol in apostrophes.

text-string

Specifies text to be written to the output file. If substitution is enabled, the *text-string* can be a string symbol in apostrophes. You can send blank lines as text to an output file, as well as characters.

The .DATA command line cannot exceed 132 characters.

Examples

1. IND creates a file TEMP.DAT containing the line `THIS IS DATA`:

```
.OPEN TEMP
.DATA THIS IS DATA
.CLOSE
```

2. The .DATA directive is also useful for creating command files that execute commands requiring more than one line. The following example creates and executes a command file that runs PIP, unprotects the file DU0:FILE.TST, and copies the file to DU1:

```
.OPEN COPY.TMP
.DATA RUN PIP
.DATA DU0:FILE.TST/Z
.DATA DU1:FILE.TST=DU0:FILE.TST/W/Y
.DATA ^C
.CLOSE
$@COPY.TMP
```

.DEC

Decrements the value of a numeric symbol by one.

Format

.DEC *numeric-symbol*

where:

numeric-symbol

Specifies the numeric symbol.

IND displays an error message and terminates processing if you use a logical or string symbol with the .DEC directive.

.DELAY

Delays control-file processing for a specified period of time.

Format

.DELAY *nnu*

where:

nn

Specifies the number of time units for which you wish to delay execution.

IND interprets this number as octal, unless you use a decimal point to denote decimal.

u

Specifies one of the following time units:

T	Ticks
S	Seconds
M	Minutes
H	Hours

Description

The .DELAY directive is valid only if your monitor includes timer support and your configuration includes a clock.

When .DELAY suspends execution, IND displays the following message at the terminal:

```
Delaying...
```

When execution resumes, IND displays the following at the terminal:

```
...Continuing
```

Examples

1. The .DELAY directive delays execution for 8 seconds (10 octal) if .ENABLE OCTAL (the default) is in effect:

   ```
   .DELAY 10S
   ```

2. The .DELAY directive delays execution for 25_{10} seconds:

   ```
   .DELAY 25.S
   ```

.DUMP

Displays the contents of the local, global, or special symbol table, or displays the contents of all symbol tables.

Format

.DUMP *[symbol-table]*

where:

symbol-table

Specifies the symbol table whose contents you want to display: Global, Local, or Special. If you do not specify a symbol table, IND displays the contents of all three symbol tables.

Description

IND first indicates what type of symbols will be displayed. When displaying the contents of all symbol tables, IND lists the special symbols first, followed by the global symbols and lastly the local symbols. Each line of the symbol table display is formatted as follows:

SYMBOL(TYPE): VALUE

where:

SYMBOL	Specifies the symbol name.
TYPE	Specifies one of the following types of symbols:

L	`Logical symbol`
O	`Octal numeric symbol`
D	`Decimal numeric symbol`
S	`String symbol`

VALUE	Specifies the symbol's value: T or F for a logical symbol, a number for a numeric symbol, or an ASCII string in quotation marks for a string symbol.

Local symbols are displayed in reverse order of definition; the last local symbol defined is listed first. If you use the .DUMP LOCAL directive and the local symbol table is empty, IND displays:

```
****There are no local symbols****
```

Global symbols are listed in order of definition; the first global symbol defined is listed first. If you use the .DUMP GLOBAL directive and the global symbol table is empty, IND displays:

```
****There are no global symbols****
```

IND may also display one or both of these messages when displaying all three symbol tables (the .DUMP directive with no argument). Note, however, that the special symbol table is never empty because IND special symbols are permanent symbols.

Note that you cannot place a comment after a .DUMP directive.

Example

The following example displays the contents of all three symbol tables of a control file containing a .DUMP. The name of the control file is MYIND.COM. The local symbols NAME, UNITS, and SUSPND were defined in the control-file procedure before the .DUMP directive was executed:

```
.DUMP

------------------------------------------------------------
Special symbols:

MAPPED(L): T
ALTMOD(L): F
ESCAPE(L): F
DEFAUL(L): F
RAD50 (L): F
ALPHAN(L): F
EOF   (L): F
FALSE (L): F
TIMOUT(L): F
TRUE  (L): T
MODE  (L): T
OCTAL (L): T
SPACE (O): 7666
SYMTYP(O): 0
FILERR(O): 0
STRLEN(O): 1
SYUNIT(O): 0
EXSTAT(O): 1
SUCCES(O): 1
WARNIN(O): 0
ERROR (O): 2
SEVERE(O): 4
JOBS  (O): 10
SYSTEM(O): 10
MONNAM(S): "RT11XM"
SYDISK(S): "DU"
DATE  (S): "20-MAR-91"
TIME  (S): "00:06:14"
FILSPC(S): "MYIND.COM"
EXSTRI(S): ""

Global symbols:

                 ****There are no global symbols****
```

```
Local symbols:

NAME(S)  : "XYZ CORPORATION"
UNITS(O) : 4
SUSPND(L): T
P9(S)    : ""
P8(S)    : ""
P7(S)    : ""
P6(S)    : ""
P5(S)    : ""
P4(S)    : ""
P3(S)    : ""
P2(S)    : ""
P1(S)    : ""
P0(S)    : "MYIND.COM"
COMMAN(S): "MYIND.COM"
------------------------------------------------------------
```

.ENABLE/.DISABLE

Enables (.ENABLE) or disables (.DISABLE) one or more IND operating modes.

Format

.ENABLE *operating-mode[,operating-mode,...]*

.DISABLE *operating-mode[,operating-mode,...]*

where:

operating-mode

Specifies one of the following:

```
ABORT        DELETE      MCR        SUBSTITUTION    TYPEAHEAD
CONTROL-Z    ESCAPE      OCTAL      SUFFIX
DATA         GLOBAL      PREFIX     TIMEOUT
DCL          LOWERCASE   QUIET      TRACE
```

Each mode is independent of the others, and all can be active simultaneously. See the following description section for individual descriptions of each mode.

Enabling or Disabling Several Modes at a Time

Each operating mode is independent of the others, though you can enable/disable more than one operating mode at a time. To do so, separate the operating-mode parameters in the command line by commas, with one exception.

The DATA operating mode is the exception, and is the only mode that must be enabled/disabled on a line by itself. This means both the .ENABLE DATA directive and the .DISABLE DATA directive must each be alone on a command line.

In the following example, the LOWERCASE and TIMEOUT operating modes are enabled with one directive, while the SUBSTITUTION and GLOBAL operating modes are disabled with another directive:

```
.ENABLE LOWERCASE,TIMEOUT
.DISABLE SUBSTITUTION,GLOBAL
```

If you enable more than one operating mode on one line (that is, with one .ENABLE directive) and an error occurs, none of the operating modes is enabled. Also, if you disable more than one operating mode on one line (that is, with one .DISABLE directive) and an error occurs, none of the operating modes is disabled.

The .ENABLE and .DISABLE DATA Directives

You cannot specify more than one operating mode with the .ENABLE DATA or the .DISABLE DATA directive. For example, the following control file line is invalid:

```
.ENABLE DATA, OCTAL
```

Description

Operating modes can be local or global in scope. A local operating mode is enabled (disabled) in the control file in which it is defined. A global operating mode is enabled (disabled) in any control file invoked from the calling control file containing the .ENABLE/.DISABLE directive.

The Scope of Operating Modes

- Local operating modes are enabled/disabled only within the control file in which they are enabled. These modes automatically return to their default settings when you enter a nested control file. So, if the mode you enabled or disabled in a control file is a local one, you must explicitly enable or disable it in each file you nest to continue the same type of mode in that file. When you return from a nested control file to the previous level file, the local operating modes return to the settings of that previous level.
- Global operating modes remain enabled or disabled throughout all levels of control files until you explicitly change the settings.

Summary of Operating Modes

Table 4–3 alphabetically lists the operating modes with the default setting, scope, and description of each. See the individual mode descriptions following the table for more information.

Table 4–3: IND Operating-Mode Summary

Mode	Default	Scope	Description
ABORT	Enabled	Global	Allows you to abort a control file with a single CTRL/C if it is waiting for a response, with a double CTRL/C if it is running, or with a single CTRL/Z if it is waiting for a response to an .ASK, .ASKN, or .ASKS prompt.
CONTROL-Z	Enabled	Global	Allows you to abort a control file with a CTRL/Z typed in response to an .ASK, .ASKN, or .ASKS prompt.
DATA	Disabled	Local	Allows you to send more than one line of text at a time to an output line.
DCL	Enabled	Local	Allows you to use DCL or CCL commands in control files.

Table 4–3 (Cont.): IND Operating-Mode Summary

Mode	Default	Scope	Description
DELETE	Disabled	Local	Causes the control file to be deleted when its processing is completed.
ESCAPE	Disabled	Global	Allows you to use the ESCAPE character as a valid response to an .ASK, .ASKN, or .ASKS prompt.
GLOBAL	Disabled	Global	Causes all symbols beginning with a dollar sign ($) to be global to all nested control files.
LOWERCASE	Enabled	Global	Causes IND to store string characters in the case you type them.
MCR	Enabled	Local	Causes IND to pass lines it does not recognize to the keyboard monitor to be executed.
OCTAL	Enabled	Global	Causes IND to interpret numeric symbols and .ASKN responses as octal.
PREFIX	Enabled	Global	Causes IND to display an asterisk and a space in front of all .ASK, .ASKN, and .ASKS prompts and to display a semicolon in front of external comment lines.
QUIET	Disabled	Local	Suppresses the display of DCL or CCL command lines when a control file is processed.
SUBSTITUTION	Enabled	Global	Replaces symbols with their values.
SUFFIX	Enabled	Global	Causes IND to display a question mark after all .ASK prompts and to display the response specifications after all .ASK, .ASKN, and .ASKS prompts.
TIMEOUT	Disabled	Global	Allows you to use timeout counts with .ASK directives.
TRACE	Disabled	Local	Causes IND to display on the terminal each control-file command as it is processing.
TYPEAHEAD	Enabled	Global	Allows you to type a reponse before being prompted by an .ASK, .ASKN, or .ASKS prompt.

Operating Modes

ABORT

Allows you to abort a control file in either of two ways:

- By typing CTRL/C twice.
- By typing CTRL/Z in response to an .ASK, .ASKN, or .ASKS directive. See the CONTROL-Z mode description for more information on aborting a control file with CTRL/Z.

The IND directives .ENABLE ABORT and .DISABLE ABORT enable and disable ABORT mode. The default setting is .ENABLE ABORT.

If you disable ABORT mode (.DISABLE ABORT), IND ignores CTRL/C characters typed at the terminal. ABORT mode remains disabled until the currently executing control file or control file nested within it exits, or until you issue the .ENABLE ABORT directive. The automatic installation procedure disables ABORT mode as a safety measure to ensure a successful installation.

.DISABLE ABORT is available only if you have included global .SCCA support in your monitor through the system generation process. If you issue the .DISABLE ABORT directive and your monitor does not include global .SCCA support, IND returns the value 0 (for warning) in the special symbol <EXSTAT> when the directive is processed.

In the following example, double CTRL/C aborts are disabled during an INITIALIZE operation:

```
.ASKS DEV What disk do you want to initialize?
.IF DEV EQ "" .GOTO 10
.;Disable abort so user cannot abort INITIALIZE
.;with double CTRL/C
.DISABLE ABORT
INITIALIZE/BADBLOCKS/NOQUERY 'DEV':
.ENABLE ABORT

.10:
.STOP
```

CONTROL-Z

Allows you to abort a control file by typing CTRL/Z in response to an .ASK, .ASKN, or .ASKS prompt.

The IND directives .ENABLE CONTROL-Z and .DISABLE CONTROL-Z enable and disable CONTROL-Z mode. If you disable CTRL/Z (.DISABLE CONTROL-Z), IND ignores CTRL/Z typed at the terminal. CTRL/Z remains disabled until the currently executing control file (including control files nested within it) exits or until you issue the .ENABLE CONTROL-Z directive. The default setting is .ENABLE CONTROL-Z.

DATA [n]

Allows you to append more than one line of text at a time to an output file. When you use .ENABLE DATA, blank lines are ignored; this means you can use blank lines to format your control file without the blank lines being copied as data.

Use the .DATA directive whenever you want to append only one line of text to a file. Use .ENABLE DATA when you want to append many lines of text.

In the .ENABLE DATA [n] directive, *n* specifies an optional file number in the range 0–3. The default file number is 0. If substitution is enabled, you can substitute a symbol for the value *n* by enclosing the symbol in apostrophes.

Examples

- In this example, IND writes the lines that fall between the .ENABLE and .DISABLE directives to the file SECFIL.DAT:

  ```
  .OPEN SECFIL.DAT
  .ENABLE DATA
       .
       .
       .
  .DISABLE DATA
  ```

- DATA mode is useful for creating control files that execute commands using more than one line. The next example control file does that; the file runs PIP, unprotects the file DU:FILE.TST, and copies the file to DU1:

  ```
  .OPEN COPY.TMP
  .ENABLE DATA
  RUN PIP
  DU:FILE.TST/Z
  DU:FILE.TST=DU1:FILE.TST/W/Y
  ^C
  .DISABLE DATA
  .CLOSE
  $@COPY.TMP
  ```

DCL

Causes IND to pass lines it does not recognize to the keyboard monitor to be executed.

If DCL is enabled, the keyboard command ASSIGN DU0: LOG in this example is executed. If DCL is disabled, or if the /N option was used, this DCL command is ignored:

```
.ASKS DEV WHICH DEVICE WILL YOU USE FOR THE LOG FILE?
ASSIGN 'DEV': LOG
```

DELETE

Causes IND to delete the control file (in which the DELETE mode is enabled) when IND is through processing it. Processing is complete when IND executes the .EXIT directive or reaches the end of the control file.

The .ENABLE DELETE directive has the same effect as the /D option.

ESCAPE

Allows you to use the escape character as a valid response to an .ASK, .ASKN, or .ASKS directive.

- An .ASK, .ASKN, or .ASKS question answered with a single escape ESC sets the special logical symbol <ESCAPE> to TRUE.
- An .ASK question answered with an ESC RETURN sequence sets the specified logical symbol to true if the symbol has not been previously defined; otherwise, it remains unchanged.

- If you type one or more characters before or after the ESC (other than a single RETURN after the ESC), IND displays the following error message and then repeats the query:

  ```
  ?IND-E-Invalid Answer or Terminator
  ```

Examples

- The following control file contains an .ENABLE ESCAPE directive:

  ```
  ;IF YOU WANT A LIST OF OPTIONS, TYPE <ESC> <RET>
  .ENABLE ESCAPE
  .ASKS A ENTER OPTION
  .IFT <ESCAPE> .GOTO LIST
    .
    .
    .
  .LIST: ;OPTIONS ARE: A (ADD), S (SUBTRACT), D (DIVIDE)
  ```

- If you type an ESC RETURN sequence in response to the ENTER OPTION prompt displayed by the preceding control file, you get the following interaction at the terminal:

  ```
  ;IF YOU WANT A LIST OF OPTIONS, TYPE <ESC> <RET>
  * ENTER OPTION [S]: ESC RET
  ;OPTIONS ARE: A (ADD), S (SUBTRACT), D (DIVIDE)
  ```

GLOBAL

Causes symbol names beginning with a dollar sign ($) to be defined as global to all levels of control files; once such a symbol has been defined, all levels recognize it. Symbols that do not begin with a dollar sign are local to the level that defines them.

The file LORRAN.COM contains the following lines:

```
.ENABLE SUBSTITUTION
;'$X'
```

When the control file contains:

```
.ENABLE GLOBAL
.SETS $X "TEST"
@LORRAN.COM
```

IND displays on the terminal:

```
;TEST
```

LOWERCASE

Causes IND to store string symbol definitions and string responses in the case in which they are typed (uppercase characters are stored in uppercase, lowercase characters are stored in lowercase). This means strings that define symbols with the .SETS directive and responses to .ASKS directive prompts are stored in the case in which they are typed.

When LOWERCASE mode is disabled, IND stores all characters as uppercase characters, regardless of whether they were typed as uppercase or lowercase characters.

Note

- Character case is significant when comparing strings; the .IF and .TEST directives discriminate between lowercase and uppercase characters, regardless of whether LOWERCASE mode is enabled or disabled.
- If LOWERCASE mode is disabled and the response to a query is in lowercase, the special logical symbol <ALPHAN> is set to FALSE.

When the control file contains:

```
.ENABLE SUBSTITUTION,LOWERCASE
.ASKS A DEFINE STRING SYMBOL A
;'A'
```

IND displays at the terminal:

```
* DEFINE STRING SYMBOL A [S]: SQRT Subroutine
;SQRT Subroutine
```

MCR

Causes IND to pass lines it does not recognize to the keyboard monitor to be executed. When the control file contains:

```
.ASKS DEV DEVICE TO USE FOR LOG FILE? DU0:
ASSIGN 'DEV': LOG
```

and MCR mode is enabled, the DCL command ASSIGN DU0: LOG is executed. If MCR mode is disabled, or if the /N option was used, this command is ignored.

OCTAL

Causes IND to interpret numeric symbols and .ASKN directive responses as octal rather than decimal. For example, if OCTAL mode is enabled and the control file contains the line:

```
.ASKN VECTR ENTER VECTOR ADDRESS OF FIRST CONTROLLER
```

IND displays the following and interprets the response to be an octal number:

```
* ENTER VECTOR ADDRESS OF FIRST CONTROLLER [O]:
```

The .ENABLE OCTAL directive can be overridden by specifying decimal numbers in the range specification or by issuing the .DISABLE OCTAL directive.

PREFIX

Causes IND to display:

- An asterisk (*) and a space in front of all prompts resulting from .ASK, .ASKN, and .ASKS directives.
- A semicolon (;) in front of all external comment lines.

For example, suppose a control file contains the following lines:

```
.ENABLE PREFIX
.ASK CONT DO YOU WANT TO CONTINUE?
.IFF CONT .GOTO SUB2             .;DO NOT CONTINUE
;CONTINUE
```

When IND processes these lines, IND displays the following on the terminal:

```
* DO YOU WANT TO CONTINUE? [Y/N D:N]: Y RET
;CONTINUE
```

If you disable PREFIX mode, IND displays the same lines but without the asterisk-space combination and semicolon.

QUIET

Causes IND to suppress the display of keyboard command lines. The command lines are executed normally, and if they return a message or display, it is displayed on the terminal.

When the control file contains the following lines and has an affirmative response from the user, IND processes the ASSIGN command but does not display it on the terminal:

```
.ASK QUIET DO YOU WANT COMMAND LINES SUPPRESSED
.IFT QUIET .ENABLE QUIET
.IFF QUIET .DISABLE QUIET
ASSIGN DU1 OUT
```

When IND is processing a control file and you type CTRL/O, IND suppresses terminal output until it encounters a .DISABLE QUIET directive.

SUBSTITUTION

Causes IND to replace a symbol with its assigned value. The symbol must be enclosed by apostrophes. For example, if the string symbol A has been assigned the string value THIS IS A TEST, then every occurrence of 'A' will be replaced by THIS IS A TEST. When SUBSTITUTION mode is enabled, IND performs substitutions on each line before scanning the line for directives and keyboard commands.

When the control file contains:

```
.ENABLE SUBSTITUTION
.ASKS FIL SPECIFY SOURCE FILE
MACRO 'FIL'
```

IND displays at the terminal:

```
* SPECIFY SOURCE FILE [S]:SOURCE
.MACRO SOURCE
```

SUFFIX

Causes IND to display:

- A question mark after all .ASK prompts.

- Response specifications after all prompts that result from .ASK, .ASKN, and .ASKS directives.

For example, suppose a control file contains the following line:

```
.ASKN [1:9.:1] INIT NO. OF DISKS TO INIT?
```

If SUFFIX mode is enabled, IND displays the following on the terminal:

```
* NO. OF DISKS TO INIT? [D R:1-9 D:1]
```

If SUFFIX mode is disabled, IND displays only the following:

```
* NO. OF DISKS TO INIT?
```

Even when SUFFIX mode is disabled, the response specifications are still used to check the validity of the response.

TIMEOUT

Causes IND to recognize timeout counts used with .ASK, .ASKN, and .ASKS directives, if the monitor includes timer support and the configuration includes a system clock.

If the monitor does not include timer support or your configuration lacks a system clock, and the .ENABLE TIMEOUT directive is processed, IND assigns the special symbol <EXSTAT> the value 0, for warning, instead of displaying an error message. If TIMEOUT mode is disabled, timeout counts are ignored.

If the control file contains the following line and TIMEOUT mode is enabled, IND waits 15_{10} ticks for a response before using the specified default response VT100:

```
.ASKS [::"VT100":15.T] TERM CONSOLE TYPE BEING USED?
```

If TIMEOUT mode is disabled or the monitor does not include timer support, you must enter a response or press RETURN to proceed.

TRACE

Causes IND to display on the terminal each line in a control file as the control line is processed. IND displays an exclamation mark (!) before each control line containing IND directives. No leading characters are placed before lines containing only DCL or CCL commands or comments.

When TRACE mode is enabled, the effect is the same as using the /T option in the CSI command string.

TYPEAHEAD

Causes IND to use the characters you type before an .ASK, .ASKN, or .ASKS prompt in the response. If you disable TYPEAHEAD mode (.DISABLE TYPEAHEAD), IND discards characters that have been typed before processing .ASK, ASKN, and .ASKS directives. So, if you respond to an anticipated prompt (before IND displays the prompt), the response is discarded. The default setting is .ENABLE TYPEAHEAD.

In the following example, TYPEAHEAD is disabled until the directory operation is complete. Therefore, any characters typed before the .ASKS prompt is displayed are discarded:

```
.DISABLE TYPEAHEAD
$DIR DU1:
.;Since typeahead is disabled, an answer to the following
.;question will not be accepted until the directory
.;listing has finished.
.ENABLE TYPEAHEAD
.ASKS FILE What file on DU1 do you want printed?
$PRINT 'FILE'
```

.END

Marks the end of a begin-end block.

Format

.END

Description

Anything that follows an .END directive on the same line is ignored. If IND encounters more .END directives than .BEGIN directives, IND displays an error message. See the description of the .BEGIN directive for more information about begin-end blocks.

.ERASE

Deletes local or global symbol definitions from the IND symbol table.

Format

$$\textbf{.ERASE} \left\{ \begin{array}{l} \textbf{LOCAL} \\ \textbf{GLOBAL} \end{array} \right\} \quad [\textit{symbol}]$$

where:

symbol
Specifies the symbol you want to erase from the specified symbol table. If you do not specify a symbol, all symbols from that symbol table are erased.

Description

When you define a symbol, either locally or globally, IND creates an entry in its symbol table. The .ERASE directive erases either all entries in the specified table or specific entries.

IND permits you to redefine global and local symbols after you have used the .ERASE directive.

Use the .DUMP directive to see which symbols each symbol table contains. See the description of the .DUMP directive for more information.

Using .ERASE LOCAL

When you use .ERASE LOCAL without a symbol name, the IND internal local symbols P0 through P9 and COMMAN are erased as well as the local symbols that you have defined.

An .ERASE LOCAL directive outside a begin-end block erases all local symbols. An .ERASE LOCAL directive within a begin-end block erases only those local symbols defined in that block.

Using .ERASE GLOBAL

An .ERASE GLOBAL, either outside or within a begin-end block, erases all global symbols. IND reserved symbols, though, cannot be erased.

Examples

1. The .ERASE directive deletes the symbol DEV from the global symbol table:

```
.ERASE GLOBAL DEV
```

2. The .ERASE directive deletes all global symbols from the global symbol table:

```
.ERASE GLOBAL
```

.EXIT

Terminates processing of the current control file or begin-end block and returns control to the previous level control file or begin-end block.

Format

.EXIT *[value]*

where:

value

Specifies an optional numeric expression or a value for the special symbol <EXSTAT>.

Description

If the .EXIT directive is encountered in the top-level control file of a series of nested control files, IND exits and passes control to the keyboard monitor.

When IND reaches the end of a control file, the effect is the same as executing an .EXIT directive.

Note that you cannot place a comment after the .EXIT directive.

Example

Consider the following line in control file FILE1:

```
@FILE2
```

If the file FILE2.COM contains the line:

```
.EXIT
```

when IND encounters the .EXIT directive in FILE2, control returns to FILE1.COM.

If the .EXIT directive in FILE2.COM includes a numeric expression as in the following example, IND evaluates that expression and then assigns the value to <EXSTAT>:

```
.EXIT N+2
```

.GOSUB

Saves the current location in a control file and then branches to another location, identified by a label.

Format

.GOSUB *label*

where:

label

Specifies the subroutine entry point. The label must not include the leading period and trailing colon.

Description

IND can branch to any subroutine in the current control file, regardless of begin-end blocks. The maximum nesting depth for subroutine calls is eight.

To return from a subroutine to the calling location, use the .RETURN directive. See the description of that directive for more information.

Example

The following .GOSUB directive transfers control to the subroutine labeled .EVAL:

```
.GOSUB EVAL
```

.GOTO

Unconditionally transfers control to the line specified by the label argument.

Format

.GOTO *label*

where:

label

Specifies the label of the line to which you want control-file execution to branch.

The label must not include the leading period and trailing colon.

Description

Branches can go forward or backward in a control file, and all lines between a .GOTO directive and the specified label are ignored.

.GOTO Directives in Begin-End Blocks

- If a .GOTO directive appears in a begin-end block, the labeled line must be in that same block. The .GOTO directive cannot branch to a nested begin-end block, but can branch to another location in the control file that appears after a nested begin-end block.
- When IND encounters a .GOTO directive within a begin-end block, it scans the current begin-end block from top to bottom for the label within that block. Since the label scan starts at the .BEGIN directive and continues to the .END directive, labels that have multiple definitions are permitted within a block. IND finds the first definition of the label and branches to that location. However, having different labels of the same name could cause confusion and is not recommended.

Example

The .GOTO directive transfers control to the entry point labeled 100:

```
.GOTO 100
   .
   .
   .
.100:
```

.IF

Compares a *numeric* or *string* symbol with another symbol of the same type to determine if a condition is true. If the condition is true, IND executes the remainder of the command line.

When comparing string values, IND compares the ASCII codes of the corresponding characters. Therefore, IND can calculate less-than, greater-than, or equal-to relationships between string values. Differences between any characters, for example, even between uppercase and lowercase characters will appear, since their ASCII values are different.

Format

.IF *symbol operator expression action*

where:

symbol

Specifies a numeric or string symbol.

operator

Specifies one of the following relational operators.

Operator	Meaning
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

expression

Specifies an expression of the same symbol type as the first argument of the .IF directive.

action

Specifies how the processing is to continue if the test results in a true value.

Examples

1. IND compares the ASCII numeric values of two string symbols, X and Y. Since the ASCII value of string-symbol X is less than the ASCII value of string-symbol Y, control passes to the line labeled 200:

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

2. IND compares the ASCII numeric values of two numeric symbols, N1 and N2. Since the value of N1 is less than or equal to N2, IND will increment N1:

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

3. IND compares the ASCII numeric value of S1 with the ASCII numeric value of S2 concatenated with the first character of S3. The ASCII decimal value of lowercase b is 98, while the ASCII decimal value of uppercase B is 66. This means the following .IF condition is true and IND increments the value of N:

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

.IFDF/.IFNDF

Tests whether a logical, numeric, or string symbol has been *defined* or not defined. If the test is true, IND processes the remainder of the command line. These directives do not evaluate symbols.

Format

.IFDF *symbol action*

.IFNDF *symbol action*

where:

symbol
Specifies a 1- to 6-character symbol.

action
Specifies what is to be done if the specified condition is true.

Example

If symbol A is defined, control branches to the section of the control file labeled 100. If symbol A is not defined, IND displays the prompt `DO YOU WANT TO SET TIME`:

```
.IFDF A .GOTO 100
.IFNDF A .ASK A DO YOU WANT TO SET TIME
```

.IFENABLED/.IFDISABLED

Tests whether a specific operating mode is enabled (.IFENABLED) or disabled (.IFDISABLED). If the test is true, IND processes the rest of the directive line.

See the .ENABLE/.DISABLE directive for descriptions of the operating modes.

Format

.IFENABLED *operating-mode action*

.IFDISABLED *operating-mode action*

where:

operating-mode
Specifies the operating mode you want to test.

action
Specifies what is to be done if the specified condition is true.

Example

If DCL mode is enabled, execution branches to the section of the control file labeled COM. If DCL mode is disabled, execution branches to the section of the control file labeled CCL:

```
.IFENABLED DCL  .GOTO COM
.IFDISABLED DCL .GOTO CCL
```

.IFLOA/.IFNLOA

Tests whether a specific *device handler* is *loaded* or not loaded. If the test is true, IND processes the rest of the directive line.

Format

.IFLOA *device action*

.IFNLOA *device action*

where:

device
Specifies the device handler you want to test. You can use a character string or string symbol in single quotes when substitution mode is enabled to specify the device handler.

action
Specifies what is to be done if the specified condition is true.

Example

If the MU handler is loaded, control branches to the line labeled TAPE. If the MU handler is not loaded, control passes to the line labeled DISK:

```
.IFLOA MU0  .GOTO TAPE
.IFNLOA MU0 .GOTO DISK
```

.IFT/.IFF

Tests whether a *logical* symbol is true (.IFT) or false (.IFF), or tests whether specific bits in a *numeric* symbol are set to 1 (.IFT) or 0 (.IFF). See also the .AND and .OR directives for performing compound logical tests.

Format

$$\textbf{.IFT} \left\{ \begin{array}{l} \textit{logical-symbol} \\ \textit{[mask] numeric-symbol} \end{array} \right\} \textit{action}$$

$$\textbf{.IFF} \left\{ \begin{array}{l} \textit{logical-symbol} \\ \textit{[mask] numeric-symbol} \end{array} \right\} \textit{action}$$

where:

logical-symbol

Specifies a logical symbol you want to test for a true or false value.

- When you use the .IFT directive with a logical symbol, if the symbol's value is true, IND processes the remainder of the command line. If the symbol's value is false, the next command line is processed instead.
- When you use the .IFF directive with a logical symbol, if the symbol's value is false, IND processes the remainder of the command line. If the symbol's value is true, the next command line is processed.

[mask]

Specifies a numeric symbol or expression, in the range 0–177777_8 or 0–65535_{10}, that determines which bits to test for a 1 or 0 value. You must include the surrounding brackets when you use this argument.

A mask in RT–11 is a combination of bits that is used to manipulate selected portions of any word, character, byte, or register while retaining other parts for use.

- When you use the .IFT directive with [mask], IND checks to see which bits in the mask are set to 1 (for true), and tests the corresponding bits in the numeric symbol. If any of these is also set to 1, IND processes the remainder of the command line. Otherwise, the next command line is processed instead.
- When you use the .IFF directive with [mask], IND checks to see which bits in the mask are set to 1 and tests the corresponding bits in the numeric symbol. If any of these is set to 0, IND processes the remainder of the command line. Otherwise, the next command line is processed instead.

numeric-symbol

Specifies a numeric symbol whose bits you want to test for a 1 or 0 value.

action

Specifies what is to be done if the specified condition is true.

Examples

1. When these sample command lines are processed, IND branches to the part of the control file labeled 100:

   ```
   .SETT LOGA
   .IFT LOGA .GOTO 100
   ```

2. These two command lines can cause the same action as the first example, making IND branch to the part of the control file labeled 100:

   ```
   .SETT [5] NUMB
   .IFT [7] NUMB .GOTO 100
   ```

   In the first line of code, we use a filter or *mask* to set the value in `NUMB`. This filter is evaluated in the current radix.

   The filter is the binary form of the number 5, which is 101_8. So, the line `.SETT [5] NUMB` means that the first and third bytes in the binary value for `NUMB` are both set to 1. Since we may be changing the value of these two bytes in `NUMB`, these two bytes are used only as a filter. The other bytes were not affected. They contain information we want to keep.

   The binary form of the decimal number 7 is 111_8. Since the first and third bytes of this binary number are the same as the first and third bytes of the decimal 5 (and now of `NUMB`), the expression `.IFT [7] NUMB` is true and execution transfers to the line labeled 100.

3. When these sample command lines are processed, IND branches to the part of the control file labeled 100:

   ```
   .SETF LOGA
   .IFF LOGA .GOTO 100
   ```

4. These two command lines can cause the same action as the preceding example, making IND branch to the part of the control file labeled 100:

   ```
   .SETF [5] NUMB
   .IFF [7] NUMB .GOTO 100
   ```

   In the first line of code, we use a filter or *mask* to set the value in `NUMB`. The filter is the binary form of the number 5, which is 101_8. So, the line `.SETF [5] NUMB` means that the first and third bytes in the binary value for `NUMB` are both set to 0. Since we may be changing the value of these two bytes in `NUMB`, these two bytes are used only as a filter. The other bytes were not affected. They contain information we want to keep.

   The binary form of the decimal number 7 is 111_8. Since the first and third bytes of this binary number are 1, and the first and third bytes of the binary code for `NUMB` are 0, the expression `.IFF [7] NUMB` is true and execution transfers to the line labeled 100.

.INC

Adds one to a numeric symbol.

Format

.INC *numeric-symbol*

where:

numeric-symbol

Specifies the numeric symbol being incremented.

If you use the .INC directive to increment a logical or string symbol, IND displays an error message and exits from processing.

Example

IND increments numeric symbol UNITS by one:

```
.INC UNITS
```

.ONERR

Causes IND to continue processing at another location in a control file when IND detects any of the errors listed below.

Format

.ONERR *label*

where:

label

Specifies a label in a control file marking the location at which you want to continue processing. If you do not specify a label, the .ONERR directive is disabled.

Description

Usually when IND detects a fatal error, IND displays the appropriate error message and stops executing the control file. However, when you use .ONERR and one of the following fatal errors occurs, IND branches to the label you specified and continues execution there.

Errors Intercepted by the .ONERR Directive

You can use the .ONERR directive only with the errors in the following list. These are the only errors that the .ONERR directive can intercept. See Part III of this manual for explanations of how to correct the errors.

These errors are alphabetically listed according to the messages IND displays when it encounters them:

?IND–F–Bad range or default specification

?IND–F–Data file error

?IND–F–Data file open

?IND–F–Deleting special symbol

?IND–F–Device is attached

?IND–F–Error reading from terminal

?IND–F–File already open

?IND–F–File not open

?IND–F–File read error

?IND–F–Invalid attempt to erase symbol

?IND–F–Invalid file number

?IND–F–Invalid keyword

?IND–F–Invalid operator for operation

?IND–F–Invalid nesting

?IND–F–Symbol table overflow <symbol>

?IND–F–Label not at beginning of line

?IND–F–Null control string (to .PARSE directive)

?IND–F–Numeric under or overflow

?IND–F–Redefining symbol to different type <symbol>

?IND–F–.RETURN without .GOSUB

?IND–F–String substitution error

?IND–F–Subroutine nesting too deep

?IND–F–Swap error

?IND–F–Symbol type error <symbol>

?IND–F–Undefined label <.label>

?IND–F–Undefined symbol <symbol>

Placing .ONERR Before an Error

You can use .ONERR directives anywhere in your control file, but each time IND detects an error, control passes to the label specified by the most recently processed .ONERR directive in that control file. This means, to be effective, you must place the .ONERR directive before the location in the control file where IND detects an error for the error to be intercepted.

Note also that an .ONERR directive is effective *only* within the begin-end block or control file in which it is defined. To be effective in a nested control file, you must define it within that control file.

How Long .ONERR Remains in Effect

Once issued, an .ONERR directive remains in effect until:

- It is redefined (that is, IND finds another .ONERR directive in the control file).
- It is disabled (either through the .DISABLE directive or through processing).
- It is processed.

  Note that after IND detects an IND error and branches because of an .ONERR directive, IND disables *that* .ONERR directive. You must define another .ONERR directive to continue .ONERR error branching.

.OPEN

Opens a file for output.

Format

.OPEN *[#n] filespec*

where:

#n

Specifies an optional file number from 0 to 3. The default is 0.

The pound sign (#) preceding the *n* is part of the necessary syntax. Since a filename can begin with a numeric character, you need the # to tell IND that the following character is a file *number* rather than a file *name*.

The *n* is the file number.

- You can have up to four files open at any time. Specify a file number, from 0 to 3, with the file specification. The first file you open is number 0, the second is number 1, and so on.
- If substitution mode is enabled, you can substitute a numeric symbol for *n* by enclosing the symbol in apostrophes.

filespec

Specifies the file to be opened. (The default file type is DAT.)

Description

Four points:

- IND allows the opening of any device type for write operations. Remember, though, that you can open only one file at a time on a magtape volume.
- Use the .OPEN directive only when you wish to send data to a file; and before you open a file, make sure that the output file you specify is not protected.
- If you use the .OPEN directive and specify a file that already exists, IND deletes the original file when you subsequently use the .CLOSE directive.
- Before exiting from IND or passing execution to the keyboard monitor, you must close any open files.

Example

IND opens the file SECOUT.DAT for output:

```
.OPEN #0 SECOUT
```

.OPENA

Opens a file and appends all subsequent data to that file.

Format

.OPENA *[#n] filespec*

where:

#n

Specifies an optional file number from 0 to 3. The default is 0.

The pound sign (#) preceding the *n* is part of the necessary syntax. Since a filename can begin with a numeric character, you need the # to tell IND that the following character is a file *number* rather than a file *name*.

The *n* is the file number.

- You can have up to four files open at any time. Specify a file number, from 0 to 3, with the file specification. The first file you open is number 0, the second is number 1, and so on.
- If substitution mode is enabled, you can substitute a numeric symbol for *n* by enclosing the symbol in apostrophes.

filespec

Specifies the file to be opened. (The default file type is DAT.)

Description

Five points to follow when using this directive:

- Use this directive to send output to a file.
- Before you open a file, make sure it is not protected.
- If you use this directive with a file that does not already exist, this directive has the same effect as the .OPEN directive.
- You cannot issue the .OPENA directive to a magtape device.
- Before exiting from IND or passing execution to the keyboard monitor, you must close any open files.

Example

IND opens SECOUT.DAT and appends subsequent data to it:

```
.OPENA #0 SECOUT
```

.OPENR

Opens a file for input, though you cannot write to the file.

Format

.OPENR *[#n] filespec*

where:

#n

Specifies an optional file number. The default is 0.

The pound sign (#) in the format line preceding *n* is part of the necessary syntax. Since a filename can begin with a numeric character, you need the # to specify a file *number* rather than a file *name*.

- You can have up to four files open at any time. Specify a file number, from 0 to 3, with the file specification. The first file you open is number 0, the second is number 1, and so on.
- You can substitute a numeric symbol for *n* by enclosing the symbol in apostrophes.
- IND allows the opening of any directory-structured device for read operations (.OPENR directive).

filespec

Specifies the file to be opened.

Before exiting from IND or passing execution to the keyboard monitor, you must close any open files.

.PARSE

Divides a character string into substrings as specified by a control string and then stores the substrings in symbols.

Format

.PARSE *string "control-string" symbol1 symbol2 ...*

where:

string

Specifies the character string you wish to parse. This can be a either a string symbol or a character string. If you specify a character string, you must use quotes (") to begin and end the string.

control-string

Specifies the characters to be used as delimiters marking the end of each substring. You must include the quotes and not separate the delimiters.

symbol1 symbol2 ...

Specifies the string symbols into which you wish to store the substrings. Separate these symbols with spaces.

Examples

1. This example directive divides a *character string* into three parts with a colon and a period separating the parts:

   ```
   .PARSE "DU1:LNKLIB.OBJ" ":." DEV FILE TYPE
   ```

   In the example:

   - `"DU1:LNKLIB.OBJ"` is the character string to be parsed.
   - The `":."` is the control string, a *colon* and a *period*. The colon serves as a terminator for the first substring, and the period serves as a terminator for the second substring.
   - `DEV`, `FILE`, and `TYPE` are the string symbols into which the substrings are stored.

     The first substring `DU1` is stored in the symbol `DEV`, the second substring `LNKLIB` is stored in the symbol `FILE`, and the last substring `OBJ` is stored in the symbol `TYPE`.

2. This example directive divides a *string symbol* into six parts with a comma separating the parts:

   ```
   .PARSE MACFIL "," COM A1 A2 A3 A4 A5
   ```

In the example:

- `MACFIL` is a string symbol containing the string `COPY,FILE1.MAC,FILE2.MAC,,FILE3.MAC`.
- The control string has one delimiter, a comma (,).
- `COM`, `A1`, `A2`, `A3`, `A4`, and `A5` are the string symbols. When IND processes the preceding command line, it produces the following symbols.

Symbol	**Contents**
COM	COPY
A1	FILE1.MAC
A2	FILE2.MAC
A3	null
A4	FILE3.MAC
A5	null
<STRLEN>	5

This example illustrates the following points:

- If the number of substring symbols exceeds the number of characters (delimiters) in a control string, the last character of the control string will be repeated as a substring delimiter. The last substring symbol receives the remaining part of the character string.
- If there are more substring symbols than substrings, IND sets the additional substring symbols to null.
- The special symbol <STRLEN> contains the number of substrings processed, including explicit null symbols.

.PURGE

Discards or closes a specified output file and frees its file number, without taking any other action.

Format

.PURGE *[#n]*

where:

#n

Specifies an optional file number from 0 to 3. The default is 0.

The pound sign (#) preceding the *n* is part of the necessary syntax. Since a filename can begin with a numeric character, you need the # to tell IND that the following character is a file *number* rather than a file *name*.

The *n* is the file number. If substitution mode is enabled, you can substitute a symbol for *n* by enclosing the symbol in apostrophes.

Description

The .PURGE directive can have a different effect depending on which type of open directive you used to open the file you are purging. If you .PURGE a file previously opened with:

- .OPEN, IND discards the file. The file does not become permanent.
- .OPENA, IND makes no changes to the file. Any I/O is discarded.
- .OPENR, IND makes no changes to the file. Any I/O is discarded.

.READ

Reads an ASCII record from a file previously opened with the .OPENR directive.

Format

.READ *[#n] string-symbol*

where:

#n

Specifies an optional file number from 0 to 3. The default is 0.

The pound sign (#) preceding the *n* is part of the necessary syntax. Since a filename can begin with a numeric character, you need the # to tell IND that the following character is a file *number* rather than a file *name*.

The *n* is the file number. If substitution mode is enabled, you can substitute a symbol for *n* by enclosing the symbol in apostrophes.

string-symbol

Specifies the string symbol that is to store a record.

Description

An ASCII record is a string of characters delimited by line terminators. A string symbol stores a record, but since the string symbol (variable) cannot exceed 132 characters, file records cannot exceed 132 characters, including the return and line feed characters at the end of the record.

Reading to End-of-File and Null Characters

- When .READ encounters the end-of-file indicator, the <EOF> symbol is set. If the end-of-file has occurred and another .READ directive is issued, both <EOF> and <FILERR> will be set to indicate end-of-file.
- The .READ directive ignores null characters.
- When processing is complete, use the .CLOSE or .PURGE directive to close a file.

.RETURN

Returns control from a subroutine to the most recently saved position in the control file. Place this command at the end of a subroutine.

Format

.RETURN

.SETD/.SETO

Sets the radix of a numeric symbol to decimal (.SETD) or octal (.SETO). These directives do not alter the value of the symbol, only its radix.

Format

.SETD *numeric-symbol*

.SETO *numeric-symbol*

where:

numeric-symbol
Specifies the numeric symbol whose radix is being changed.

Example

In the following example, the value of the numeric symbol UNITS is set to 10_{10} and then changed to 12_8:

```
.SETN UNITS 10.
.SETO UNITS
```

Note

- You must use .SETN to set a numeric value. By specifying the period after the value, you automatically set that value to a decimal radix without having to use the .SETD directive.
- If, in this example, the 10 were without a period (10), its radix would have been octal by default. If you had then used the code .SETD UNITS, the 10_8 would have become an 8_{10}.

.SETL

Sets or clears the bits of a logical symbol depending on the value of a logical expression:

- If the symbol has not been defined, IND makes an entry in the symbol table and sets the symbol to the value (set or cleared) of the logical expression.
- If the symbol has already been defined, IND resets the symbol accordingly.
- If the logical symbol was previously defined as a numeric or string symbol, IND displays an error message and exits from processing.

Format

.SETL *logical-symbol logical-expression*

where:

logical-symbol

Specifies the logical symbol to be set or cleared.

logical-expression

Specifies a logical expression that can include logical symbols and numeric values joined by the three logical operators:

&	(logical AND)
!	(logical OR)
^	(logical NOT)

No embedded blanks or tabs are permitted. IND evaluates from left to right unless parentheses are used to form subexpressions, which are evaluated first. If any value in an expression is specified as decimal, IND assumes that all values in the expression are decimal; otherwise, all values are octal.

Example

The following .SETL directive sets the logical symbol MONITR equal to the expression FB!XM!ZM. This means, if any one of the three logical symbols (FB, XM, or ZM) is set to true, the logical symbol MONITR is set to true. If none of the three is set to true, MONITR is set to false:

```
.SETL MONITR FB!XM!ZM
```

.SETN

Defines or changes the *numeric* value of a specified symbol:

- If the symbol has not been defined, IND makes an entry in the symbol table and sets the symbol to the numeric value specified.
- If the symbol has already been defined, IND resets the symbol accordingly.
- If the numeric symbol was previously defined as a logical or string symbol, IND displays an error message and exits from processing.

Format

.SETN *numeric-symbol numeric-expression*

where:

numeric-symbol

Specifies a numeric symbol.

numeric-expression

Specifies a numeric expression. You can combine numeric symbols and constants to form a numeric expression, but no embedded blanks or tabs are allowed. IND evaluates from left to right unless parentheses are used to form subexpressions, which are evaluated first.

Determining the Radix of the Expression

- If octal mode is enabled, all values specified without decimal points are assumed to be octal.
- If decimal mode is enabled, all values are assumed to be octal, whether they have decimal points or not.
- All values specified with decimal points are always treated and stored as decimal values.
- If octal mode is enabled and one or more values have a decimal point, IND converts to decimal those values without decimal points, before it performs the arithmetic operation.

Examples

1. IND assigns the integer 27_8 to the numeric symbol NUMBER:

   ```
   .SETN NUMBER 27
   ```

2. IND assigns the value of the numeric expression (the value of symbol A2 minus 5, multiplied by 3) to the numeric symbol A1:

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

.SETS

Defines or changes the value of a specified string symbol:

- If the symbol has not been defined, IND makes an entry in the symbol table and sets the symbol to the specified string value.
- If the symbol has been defined, IND resets the symbol accordingly.
- If the symbol has been defined previously as logical or numeric, IND displays an error message.

Format

.SETS *string-symbol* *string-expression*

where:

string-symbol
Specifies a string symbol.

string-expression
Specifies any string expression.

To form a string expression, you can combine with the plus signs (+) a string symbol, constant, or substring with another string symbol or substring. You must enclose string constants with quotes.

Examples

1. IND assigns the string value ABCDEF to the symbol A:

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

2. IND assigns the value of the string expression STR2+"ABC" to the symbol X. Assuming the symbol STR2 contains ZZZ, the resulting value is ZZZABC:

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

3. IND assigns the value of the string expression STR2+A[4:6] to the symbol X. Assuming the symbol STR2 contains ZZZ, and assuming string A contains ABCDEF, the resulting value is ZZZDEF. The [4:6] means *take the substring from the fourth character through the sixth character.* The fourth through sixth character in string A is DEF:

   ```
   .SETS X STR2+A[4:6]
   ```

4. IND assigns the string expression <SYDISK>+":MYFILE.TXT" to the string symbol MYFILE. In the expression, IND combines the value of the special symbol for the system device with the string MYFILE.TXT. For example, if the system device is DU:, the resulting string value of MYFILE is DU:MYFILE.TXT:

   ```
   .SETS MYFILE <SYDISK>+":MYFILE.TXT"
   ```

.SETT/.SETF

- Assigns a value of true (.SETT) or false (.SETF) to a *logical* symbol:
  - — If the logical symbol has not been defined, IND makes an entry in the symbol table and sets the symbol to the value specified.
  - — If the logical symbol has already been defined, IND resets the symbol accordingly.
- Redefines a *previously defined numeric* symbol by setting and clearing bits. You can use these directives with a numeric symbol only to reset bits in a previously defined symbol.
- If the logical or numeric symbol you specify was previously defined as another type of symbol, IND displays an error message, and processing halts.

See also the .IFT and .IFF directives.

Format

$$\textbf{.SETT} \quad \left\{ \begin{array}{l} \textit{logical-symbol} \\ \textit{[mask]} \quad \textit{numeric-symbol} \end{array} \right\}$$

$$\textbf{.SETF} \quad \left\{ \begin{array}{l} \textit{logical-symbol} \\ \textit{[mask]} \quad \textit{numeric-symbol} \end{array} \right\}$$

where:

logical-symbol

Specifies a logical symbol you want to define or redefine.

[mask]

Specifies a numeric symbol or expression, in the range 0–177777_8 or 0–65535_{10}, that determines which bits are to be set or cleared in a numeric symbol. You must include the surrounding brackets when you use this argument.

- When you use the .SETT directive with [mask], for every bit in the mask that is set to 1, the corresponding bit in the numeric symbol is also set to 1. For every bit in the mask that is set to 0, the corresponding bit in the symbol remains unchanged.
- When you use the .SETF directive with [mask], for every bit in the mask that is set to 1, the corresponding bit in the numeric symbol is set to 0. For every bit in the mask that is set to 0, the corresponding bit in the symbol remains unchanged.

numeric-symbol

Specifies a numeric symbol to be redefined by setting or clearing bits as dictated by the mask.

Examples

1. IND sets the logical symbol X to true:

   ```
   .SETT X
   ```

   As a result of this .SETT directive, X can be used in the following command line:

   ```
   .IFT X .GOTO 300
   ```

2. The binary value for the number 7 is 0111_8. So, when IND uses that code as a mask, IND sets the first three bits of the previously defined numeric symbol NUM to 1. The rest of the bits remain as they were previously defined:

   ```
   .SETT [7] NUM
   ```

   As a result of this .SETT directive, when IND processes the following command line, control branches to the line having the label SUB1:

   ```
   .IFT [7] NUM .GOTO SUB1
   ```

3. IND sets the logical symbol LOG to false:

   ```
   .SETF LOG
   ```

4. The binary value for the number 12_{10} is 1100. So, when IND uses that code as a mask, IND sets the third and fourth bits of the numeric symbol NUM to 0. The rest of the bits remain as they were previously defined:

   ```
   .SETF [12.] NUM
   ```

   As a result, when IND processes the following command line, control branches to the label SUB1. This is because the third and fourth bits of NUM are compared with the third and fourth bits of the binary value for the number 22_{10}. Since the binary value for the number 22_{10} is 010010, the fourth bit for that number is 1. And, since the fourth bit for NUM is set to 0, these two fourth bits are not alike:

   ```
   .IFF [22.] NUM .GOTO SUB1
   ```

.STOP

Halts control file processing and displays the following message on the terminal:

```
@ <EOF>
```

The .STOP directive has the same effect as the logical end-of-file directive (/).

Format

.STOP

.STRUCTURE

Determines the file structure of a specified file-structured device.

Format

.STRUCTURE *numeric-symbol device*

where:

numeric-symbol

Is a string symbol in which IND will store an octal number. That number represents the file structure of the volume in the device you specify. See the following description section for the list of octal numbers that IND uses. Next to each numeric-symbol is its meaning.

device

Specifies the device containing the volume whose file structure you want to determine. You can specify the device by:

- Its physical or logical name
- A string symbol

  If you use a string symbol, you must enclose it in quotes and you must have enabled substitution mode.
- A character string

The volume in the device must conform to Digital's boot standard for IND to be able to determine its file structure. If a volume does not conform to Digital's boot standard, .STRUCTURE returns the value 000.

A colon following the device mnemonic is ignored.

Description

The following list contains the octal numbers IND uses for the *numeric-symbol*. Next to each octal number is the value of that number (what it represents):

Octal Number	Value
000	Unknown
001–007	Reserved
010–017	PDP–8 (12-bit systems)
010	WPS–8
011	OS–8
012	COS

Octal Number	Value
020–027	RT–11 systems
020	RT–11 V5
030–037	RSTS systems
030	RSTS V8
040	Reserved
041–047	FILES–11 systems
041	FILES–11 Level 1
042	FILES–11 Level 2
050–057	UNIX* systems
050	UNIX*
060–067	DSM systems
060	DSM V3
070–077	Reserved
100–107	CP/M systems
100	CP/M
110–117	UCSD p–systems†
110	UCSD p–system†
120–127	MS–DOS systems
120	MS–DOS
130–177	Reserved
200–377	CSS and customer file structures

*UNIX is a registered trademark of American Telephone & Telegraph Company.
†USCD p–system is a trademark of University of California Regents.

Example

The following example returns 020 in the symbol SYSDSK if DW contains an RT–11 V5 disk:

```
.STRUCTURE SYSDSK DW:
;'SYSDSK'
.EXIT
```

.TEST

Tests and finds the:

- Type (string or numeric) of a symbol.
- Type (alphanumeric or RAD50) of characters in a string symbol.
- Starting position of an ASCII string within a character string.
- Radix of a numeric symbol.

Format

$$\textbf{.TEST} \quad \left\{ \begin{array}{l} \textit{symbol} \\ \textit{string-symbol} \quad \textit{[match-string]} \end{array} \right\}$$

where:

symbol
Specifies the symbol you want to test.

string-symbol
Specifies the string symbol you want to test.

match-string
Specifies the optional ASCII string whose starting position within a character string you want to find. The *match-string* variable can represent an ASCII string, a string symbol, or an expression.

Description

Searching for a Symbol Type

To search for a symbol type, use the .TEST *symbol* format. When you do this, IND indicates the symbol's type by storing the proper numeric code in <SYMTYP>.

- If the symbol is found to be a string symbol, IND sets the special symbols <ALPHAN>, <RAD50>, and <STRLEN> accordingly.
- If the symbol is found to be a numeric symbol, IND reports the radix of the symbol by setting the special symbol <OCTAL> to true if the symbol's radix is octal or to false if the symbol's radix is decimal.

Searching for the Start of an ASCII String

To search for an ASCII string within a character string, use the .TEST *string-symbol match-string* format. Then test the contents of <STRLEN>.

- If <STRLEN> has a value of 0, the match string you specified was not found in the character string.
- A nonzero value in <STRLEN> specifies the position of the match string in the character string.

Examples

1. In this example, IND:

   - Sets the string symbol ADDR equal to the word "STREET".
   - Tests to see if the string "STREET" is stored in the ADDR symbol and if it is, at what character position within the symbol that string starts.
   - Tests the special symbol <STRLEN> for the position of the match string "STREET" within the ADDR string symbol since the results of the previous two tests are stored in <STRLEN>.

   In this case, since the string "STREET" begins on the first character in the string represented by ADDR, <STRLEN> will have a value of 1.

   Note that the match string is found within the target string only if both are in uppercase characters or both are in lowercase characters:

   ```
   .SETS ADDR "STREET"
   .TEST ADDR "STREET"            ;Look for "STREET"
                                  ;in string contained in
                                  ;string symbol ADDR
   .IF <STRLEN> = 0 .GOTO NOTFND  ;String not found
   .IF <STRLEN> <> 0 .GOTO POS    ;Starting position of string
   ```

2. In this example, IND:

   - Enters the number of characters in the string symbol A into <STRLEN>.
   - Sets <ALPHAN> and <RAD50> accordingly.

   This test is useful since it makes the special numeric symbol <STRLEN> available to compare the length of string symbol A to a numeric constant or expression:

   ```
   .TEST A
   ```

3. In this example, IND tests the numeric symbol UNITS for a numeric radix. Since the range specification contains a decimal point after the number 10, the response will be interpreted as decimal. Therefore, when IND tests the symbol UNITS, the special symbol <OCTAL> will be false because the radix of the symbol is decimal:

   ```
   .ASKN [1:10.] UNITS  OF UNITS ERROR LOGGER SUPPORTS?
   .ASKN [1:7] DEVSLT  EXTRA DEVICE SLOTS WANTED?
   .IF DEVSLT > 1 .GOTO DEVSUB
   .TEST UNITS
   ```

.TESTDEVICE

Obtains the following information on the specified device:

- Physical device name
- Device size
- Whether the device handler is loaded
- Whether the device is on line
- Whether a logical disk has been mounted
- Whether the device is attached to a job

Format

.TESTDEVICE *device*

where:

device

Specifies the device you want to test. The colon (:) following the device name is optional.

Description

The results of the test are stored in the special symbol <EXSTRI>. You can use the .PARSE or .TEST directive to move the information in <EXSTRI> to separate string symbols for inspection.

- If the device name you specify is invalid or the device is not installed, <EXSTRI> contains the characters NSD (no such device).
- If the device is valid, the following nine fields of information are returned in <EXSTRI>.

Field	Contents
1	Physical device name. If no unit number is specified on input, unit 0 is assumed.
2	Device size, displayed as a decimal number (with a decimal point). If the device has a variable-sized handler and no volume is mounted in the drive, the size returned is the smallest for that device.
3	0
4	0
5	0

Field	Contents
6	LOD (device handler loaded) or UNL (device handler not loaded).
7	ONL (on-line), OFL (off-line), or UNK (unknown). IND checks for this attribute by attempting to read from the device. If a volume is mounted in the device and IND is able to read it, the ONL attribute is returned in field 7. If a read error occurs, such as when no volume is mounted, the OFL attribute is returned. Also, if the device tested is a logical disk, and the file assigned to the logical disk is for any reason not accessible, the OFL attribute is returned. Devices that are write-only, and sequential-access devices that are not non-RT–11 directory-structured (such as TT: and CR:), return the attribute UNK.
8	MTD (mounted) or NMT (not mounted). The MTD attribute is returned for logical disk devices that have been assigned with the keyboard monitor MOUNT command. Any device not assigned with the MOUNT command, or any device that is not a logical disk, returns NMT in field 8.
9	ATT (attached) or NAT (not attached). The ATT attribute is returned in <EXSTRI> if the requested device is attached to a specific job. A magtape, cassette, or any other sequential device that has a currently open file returns the ATT attribute. An attached device assumes that LOD and ONL are also true, and returns them in <EXSTRI> along with ATT. The NAT attribute is returned in <EXSTRI> if the requested device is not attached to any specific job.

In <EXSTRI>, each field of information is followed by a comma, including field 9.

Examples

1. In the following example, LD2 (logical disk unit 2) is associated with the file DU1:MASTER.DSK, and also assigned the logical-device name SEC. The example shows the contents of <EXSTRI> when the .TESTDEVICE directive is used with SEC:

```
.; DU0 IS SYSYEM DEVICE
.; LD HANDLER IS LOADED
$CREATE DU1:MASTER.DSK/ALLOC:1000
$MOUNT LD2: DU1:MASTER SEC
$INIT/NOQ SEC:
.TESTDEVICE SEC
.;LD2,1000.,0,0,0,LOD,ONL,MTD,NAT
;'<EXSTRI>'
```

The contents of <EXSTRI> show:

- The physical device name is LD2.
- The device size is 1000_{10} blocks.
- The LD handler is LOADed.

- The device on which LD2 resides (DU1) is on line.
- LD2 has been assigned with the MOUNT command.
- SEC is not attached to a job.

2. In the next example, device `DU0` is tested:

   ```
   .; DU0 IS SYSYEM DEVICE
   .TESTDEVICE DU0:
   .;LD2,1000.,0,0,0,LOD,ONL,MTD,NAT
   .;DU0,65535.,0,0,0,LOD,ONL,NMT,NAT,
   ;'<EXSTRI>'
   ```

   The contents of <EXSTRI> show:

   - The physical device name is `DU0`.
   - The device size is $65{,}535_{10}$ blocks.
   - The DU handler is LOADed.
   - The device is on line.
   - DU has not been assigned with the MOUNT command.

.TESTFILE

- Checks to see whether a specified file exists.
- Places the name of the specified file in the special symbol <FILSPC>.
- Places the results of the test in the special symbol <FILERR>.

Format

.TESTFILE *filespec*

where:

filespec
Specifies the file whose existence you wish to verify. The default device specification is DK, and the default file type is DAT.

Example

In this example, the file MASTER.DAT is verified as being on DU1:

```
.; DU0 IS SYSYEM DEVICE
$CREATE DU1:MASTER.DAT/ALLOC:100
.TESTFILE DU1:MASTER
.;DU1:MASTER.DAT
;'<FILSPC>'
.;1
;'<FILERR>'
```

.VOL

Assigns the volume identification of a volume to a string symbol.

Format

.VOL *string-symbol dev*

where:

string-symbol
Specifies the string symbol in which to store the volume identification.

dev
Specifies the device containing the volume from which to read the volume identification. You can use either a string symbol in single quotes when substitution mode is enabled or a character string, to specify the device. The colon following the device mnemonic is ignored.

Description

For IND to read the volume identification, the volume must be an RT–11 file-structured device.

IND reads only the volume identification from the specified volume and stores it in the specified symbol. The owner name is not stored. The length of the volume identification is predetermined (12 spaces). If the volume identification is less than 12 characters, the value of the symbol is padded with blank characters up to 12 spaces.

Part III
IND Error Messages

Part III describes, lists, and shows you how to respond to IND error messages.

- Chapter 5 alphabetically lists descriptions and responses to IND primary error messages.
- Chapter 6 alphabetically lists descriptions and responses to IND secondary error messages.

Chapter 5

IND Primary Error Messages

5.1 IND Error Message Format

IND has two types of error messages: primary and secondary. IND displays a primary message for all errors. IND displays both a primary and a secondary message only if a control-file error occurs during a data input/output transfer. In that case, the secondary message further defines the error condition. In all cases, IND also displays the line in the control file that caused the error.

Primary Message Format

IND error messages have the following general format:

```
?IND–ErrorLevel–MessageText
                Line In Control File
```

Secondary Message Format

IND error messages dealing with input/output transfers have the format:

```
?IND–ErrorLevel–Primary Message
                Secondary Message
                Line In Control File
```

5.2 IND Error Levels

The error level is a single letter indicating the severity of the error. This letter can be one of the following: I (information), W (warning), E (error), F (fatal), and U (unconditional abort). See the *RT–11 System Message Manual* for a full descripton of these severity levels.

You can use the monitor SET command to change the effect of the severity level on the operating system's response. In particular, these settings affect the execution of indirect command files. See the *RT–11 Commands Manual* for a description of the SET ERROR command.

5.3 IND Primary Error Message Descriptions

?IND–E–Invalid Answer or Terminator

Explanation:

1. IND encountered an invalid response to an .ASK, .ASKN, or .ASKS directive prompt. For example, an alphabetic character response may have been given to an .ASKN prompt.

2. Escape recognition is disabled, and an escape character was entered in response to a prompt. The prompt is repeated.

User Action: Make sure that the response to the .ASK, .ASKN, or .ASKS prompt is valid.

?IND–E–String length not in range

Explanation: The response entered to an .ASKS directive prompt fell outside the range designated by the prompt. The prompt is repeated.

User Action: Make sure that the response entered is within the specified range.

?IND–E–Value not in range

Explanation: The numeric value entered in response to an .ASKN directive prompt fell outside the range designated by the prompt. The prompt is repeated.

User Action: Make sure that the response entered is within the specified range.

?IND–F–Bad range or default specification

Explanation:

1. The default response fell outside the range specified with an .ASKN or an .ASKS prompt, the range used an invalid radix, or the range was specified in an invalid sequence.
2. The optional default value assigned to the symbol in the .ASKS directive is not a string literal or another string symbol.

User Action:

1. Make sure that a valid radix is used in the range specification and that the range is specified from low to high.
2. Make sure that the default value assigned to the symbol in the .ASKS directive is a string literal or another string symbol.

?IND–F–Data file error

Explanation: A hard error was encountered while IND was processing an .OPEN, .OPENA, .CLOSE, or .DATA directive or a data mode access to the output file.

This message sometimes includes a secondary error message. For a listing of the secondary message and its explanation, see Chapter 6.

User Action: See the procedures for recovery from hard error conditions listed in the *RT–11 System Message Manual*.

?IND–F–Data file open

Explanation: IND attempted to execute a keyboard command before closing open files.

User Action: Close open files before the control file exits from IND.

?IND–F–Deleting special symbol

Explanation: An attempt was made to delete an IND special symbol.

User Action: Check for a typing error. Do not try to delete an IND special symbol.

?IND–F–Error reading from terminal

Explanation: IND encountered an error while trying to receive data from the console.

This message sometimes includes a secondary error message. For a listing of the secondary message and its explanation, see Chapter 6.

User Action: See the procedures for recovery from hard error conditions listed in the *RT–11 System Message Manual*.

?IND–F–File already open

Explanation: An attempt was made to open a file with the .OPEN, .OPENA, or .OPENR directive and the file is already open.

User Action: Make sure that the file is closed before trying to open it.

?IND–F–File not found

Explanation: The control file IND tried to process does not exist in the directory of the specified volume.

User Action: Check the control file specifications to make sure that they are correct and that the specified control file exists. Also, make sure that the specified volume is mounted.

?IND–F–File not open

Explanation: An attempt was made to access a file that is not open.

User Action: Make sure that the file is open before trying to access it.

?IND–F–File read error

Explanation: A hard error occurred when IND tried to read a file.

User Action: See the procedures for recovery from hard error conditions listed in the *RT–11 System Message Manual*.

?IND–F–Invalid attempt to erase symbol

Explanation: An attempt was made to delete a symbol outside a Begin/End block.

User Action: Try the operation again. Delete symbols only within the current Begin/End block.

?IND–F–Invalid command

Explanation: IND encountered an error in a command line used to execute another control file.

User Action: Make sure that nested control file specifications and options are correct. See the *RT–11 Commands Manual* for information on nested control file specifications and options.

?IND–F–Invalid device or unit

Explanation: The device specification for a file IND tried to access is invalid, or the device specification is for a nonexistent device.

User Action: Make sure that all device specifications are valid.

?IND–F–Invalid device or unit, device is attached

Explanation: The .VOL directive was issued for a device assigned to the foreground or a system job using the LOAD ddn:=F or LOAD ddn:=job commands.

User Action: You can intercept that error by using the .ONERR directive. Use the .TESTDEVICE directive to determine if the device is attached and to what job.

?IND–F–Invalid file number

Explanation: The file number specified is outside the valid range 0 to 3.

User Action: Make sure that the file number falls within the valid range.

?IND–F–Invalid keyword

Explanation: An unrecognized keyword, preceded by a period, was specified in the command.

User Action: Make sure that all directives are spelled correctly and that the directives are in the correct syntax.

?IND–F–Invalid nesting

Explanation: IND found .END without .BEGIN or .BEGIN without .END directives in the control file.

User Action: Make sure that each .BEGIN and .END directive has a corresponding .END and .BEGIN directive.

?IND–F–Invalid operator for operation

Explanation: You attempted to use an arithmetic operator (+, –, *, or /) in a logical expression with the .SETL directive.

User Action: Do not use arithmetic operators in .SETL logical expressions. Use only the logical operators & (AND), ! (OR), and ^ (NOT).

?IND–F–Invalid option

Explanation: IND encountered an unrecognized option.

User Action: Enter the command again. Use only the valid options listed in the *RT–11 Commands Manual*.

?IND–F–Label not at beginning of line

Explanation: A label does not appear as the first character(s) on a line, except for spaces or tabs.

User Action: Make sure that the labels have no embedded spaces and that the labels do not appear in the middle of the command line.

?IND–F–Maximum indirect files exceeded

Explanation: An attempt was made to access a control file at a depth greater than three levels.

User Action: Make sure that the nesting limit of three is not exceeded. This limit does not include the initial level.

?IND–F–Null control string

Explanation: No delimiters were defined in a .PARSE directive control string. A control string cannot be null.

User Action: Make sure that the correct syntax is used for a control string.

?IND–F–Numeric under- or overflow

Explanation: A value assigned as a numeric symbol falls outside the valid range 0 to 177777 (octal) or 65535 (decimal).

User Action: Make sure that all numeric symbols have values within the valid range. Make sure that no arithmetic expression that IND tries to assign to a numeric symbol yields a result outside the valid range.

?IND–F–Prompt string too large

Explanation: The prompt issued with the .ASK, .ASKN, or .ASKS directive has too many characters.

User Action: Specify a prompt that has a valid number of characters.

?IND–F–Redefining special symbol

Explanation: An attempt was made to change the value of a special symbol.

User Action: Do not try to change the value of an IND special symbol.

?IND–F–Redefining symbol to different type <symbol>

Explanation:

1. An .ASK, .ASKN, .ASKS, .READ, .SETT, .SETF, .SETL, .SETN, or SETS directive was used in an attempt to set the specified defined symbol to a different type. The first definition of a symbol determines its type, and subsequent redefinitions must conform to the original type.
2. The numeric symbol specified with the .SETT, .SETF, or .SETL directive has not been previously defined.
3. The numeric symbol specified with the .IFT or .IFF directive has been defined as a logical or string symbol.

User Action:

1. Do not try to change the type of a defined symbol. If you redefine a symbol, make sure that the new definition has the same type as the original symbol.
2. Define the numeric symbol being used before you specify it with the .SETT, .SETF, or .SETL directive.
3. Define a new symbol as a numeric type. Specify the numeric symbol with the .IFT or .IFF directive.

?IND–F–.RETURN without .GOSUB

Explanation: A .RETURN statement does not have a corresponding .GOSUB statement.

User Action: Make sure that there are no extraneous .RETURN statements and that all .GOSUB statements are used correctly. Make sure that all .RETURN statements are located correctly in the program logic.

?IND–F–String expression exceeds limit

Explanation: The limit of 132 characters in a string expression was exceeded; a string concatenation operation yielded a string that exceeds the 132-character limit; or quotes do not appear at the end of a string expression.

User Action: Make sure that string expressions do not exceed the 132-character limit. Make sure that all string expressions are properly enclosed by quotes.

?IND–F–String substitution error

Explanation: IND encountered an error while substituting a symbol.

User Action: Make sure that all substituted symbols have been defined and that each substituted symbol is enclosed with apostrophes.

?IND–F–Subroutine nesting too deep

Explanation: The maximum subroutine nesting level of eight is exceeded.

User Action: Make sure that the maximum nesting level is not exceeded.

?IND–F–Swap error

Explanation: IND encountered an error while writing to itself.

This message sometimes includes a secondary error message. For a listing of the secondary message and its explanation, see Chapter 6.

User Action: Make sure that the system device is not write locked.

?IND–F–Symbol table overflow <symbol>

Explanation: The IND symbol table is full; there is no space for the symbol represented by <symbol>.

User Action: Use the .ERASE directive to delete symbol definitions from the symbol table.

?IND–F–Symbol type error <symbol>

Explanation: The symbol <symbol> was not used in the context for its type; for example, a numeric expression referenced a logical symbol.

User Action: Compare only symbols of the same type.

?IND–F–Syntax error

Explanation: IND encountered an unrecognizable element. Common examples of syntax errors are misspelled commands, unmatched parentheses, embedded or missing spaces, and other typographical errors.

User Action: Check for typing errors and enter the command again.

?IND–F–Undefined label <.label>

Explanation: IND did not find the label, represented by <.label>, specified in a .GOTO, .GOSUB, or .ONERR directive.

User Action: Check for a typing error. Check the program logic and insert the label, represented by <.label>, where appropriate.

?IND–F–Undefined symbol <symbol>

Explanation: A symbol in the command line is not defined.

User Action: Make sure that all symbols being used or substituted are defined.

?IND–F–Wrong version of RT–11

Explanation: An attempt was made to run an RT–11 Version 5 utility (IND) on a previous version of RT–11.

User Action: Do not run RT–11 Version 5 utilities under earlier RT–11 versions.

?IND–W–Timeout support not available

Explanation: IND tried to process the .ENABLE timeout directive on a system that does not have a clock, or while a monitor without timeout support was running.

This message sometimes includes a secondary error message. For a listing of the secondary message and its explanation, see Chapter 6.

User Action: Remove the .ENABLE timeout directive from the program, use a monitor that has timeout support, or run the program on a system that has a clock.

Chapter 6

IND Secondary Input/Output Error Messages

Bad file name

Explanation: The file specification in the command line is invalid.

User Action: Make sure that the format of the file specifications is correct.

Bad record type—not ASCII data

Explanation: The file specified as input to IND does not contain valid ASCII characters.

User Action: Make sure that there are no typing errors in the specified file.

Data overrun

Explanation: More than 80 characters were entered at the terminal, or an attempt was made to read a record that contains more than the maximum of 80 characters.

User Action: Make sure that input entered at the terminal does not exceed 80 characters.

Device full

Explanation: Not enough room is available in the directory of the output device to create the specified output file. This error generally occurs when IND attempts to execute a .OPEN or a .CLOSE directive.

User Action: Make sure that the output device has enough room for all the files to be stored there. See the *RT–11 System Message Manual* for information on how to increase storage space.

Device read error

Explanation: IND encountered a bad block or another type of hard error when trying to access a file.

User Action: Make sure that each device accessed is mounted. See the procedures for recovery from hard error conditions listed in the *RT–11 System Message Manual*.

Device write error

Explanation: IND encountered a bad block or another type of hard error when trying to send output to a file.

User Action: Make sure that the device to which data is being sent is mounted and is write enabled. See the procedures for recovery from hard error conditions listed in the *RT–11 System Message Manual*.

File accessed for read

Explanation: An attempt was made to write to a currently open file with the .OPENR directive.

User Action: Close or purge the file, then open it for output.

File accessed for write

Explanation: An attempt was made to read a currently open file with the .OPEN or .OPENA directives.

User Action: Close or purge the file, then open it for input.

File already open

Explanation: An attempt was made to open a file with a file number already in use.

User Action: Use an available file number or close a file that is already open.

File exceeds space allocated

Explanation: The volume receiving data is full.

User Action: Use the SQUEEZE command or the DUP /S option to compress the volume. See the *RT–11 System Message Manual* for information on how to increase storage space.

File protection error

Explanation: An attempt was made to delete a protected file or to create a file with the same name as a protected file that already exists.

User Action: Use the keyboard UNPROTECT command to disable a file's protected status or use a different name for the new file.

Invalid device or unit

Explanation: The device or unit specified with the .OPEN, .OPENA, or .OPENR directives is invalid.

User Action: Check for syntax errors in the device specification.

No file accessed on channel

Explanation: An internal error occurred in IND.

User Action: Try the operation again. If the error persists, send an SPR to Digital; include with the SPR an output listing or log file and a machine-readable copy of the control file(s).

No such file

Explanation: The file specified in the OPEN or DELETE command was not found.

User Action: Make sure that the file specification has the correct syntax and spelling.

Undefined error code

Explanation: IND has detected an error it cannot classify.

User Action: Submit an SPR to Digital; include with the SPR an output listing or log file and a machine-readable copy of the control file(s).

Appendix A
Sample IND Control Files

This appendix contains the following three sample IND control files.

Filename	Description
NEWDEV.COM	Initializes a device with an optional volume ID, owner name, and a maximum number of files allowed on the device.
INIT.COM	Initializes a device only if it has not been previously initialized. Can be used in a STRTxx.COM file to initialize the VM device.
GENCMD.COM	Builds a command file containing the commands you select and then executes that file. This is a fast way of performing the same function on more than one file; for example, deleting, copying, or editing similar files.

NEWDEV.COM

```
.; TITLE:  NEWDEV.COM
.; PURPOSE: Initializes a volume.
.;
.; CALLING SEQUENCE:
.;      IND NEWDEV.IND [DEV] [VOLID] [OWNER] [FILES]
.;    where the (optional) parameters are:
.;      DEV    Device (logical or physical) to be initialized
.;      VOLID  Volume identification string (no embedded spaces!)
.;      OWNER  Owner string (no embedded spaces!)
.;      FILES  Maximum number of files to put on device

.; The last parameter, the maximum number of files to go on the device,
.; is used to "fine tune" the number of directory segments.   If the
.; first three parameters (device, volid, and owner) are supplied, then
.; this procedure is assumed to be non-interactive, and these default
.; values for the files parameter will be used if none is supplied.
.; Otherwise, in interactive mode, the user will be prompted for all the
.; parameters.

.; The following line is for debugging purposes.  It should be commented
.; out until an error is discovered.

.; .ONERR DEBUG                 ; Take debug exit on error
.; PROCEDURE:
.; Initialization
   .ENABLE  QUIET
   .ENABLE  SUBSTITUTION
   .DISABLE PREFIX
   .DISABLE SUFFIX
   .DISABLE OCTAL

   .SETS VERSN "NEWDEV, VERSION 1"

.; Initialize segment size table.  Devices bigger than SEGSZN get
.; SEGBLN segments.

   .SETN SEGSZ1 12288.          ; Devices with more than 12288. blocks
   .SETN SEGBL1 31.             ; Get (by default) 31. segments
   .SETN SEGSZ2 2048.
   .SETN SEGBL2 16.
   .SETN SEGSZ3 512.
   .SETN SEGBL3 4.
   .SETN SEGSZ4 0.
   .SETN SEGBL4 1.

   .SETN SEGFIL (512.-5.)/7.    ; Number of files per (packed) segment
   .SETN EXTRA 3.               ; Number of extra entries needed
   .SETN SYSBLK 6.              ; Number of blocks before directory

   .SETS DEFOWN "User"          ; Default owner

   ;
   ;'VERSN'
   ;

.; Process parameters and assign default values.  Defer processing of p4,
.; maximum number of files, until later.

   .SETT BATCH                  ; True if parameters all supplied

   .SETS DEVICE P1

   .TEST DEVICE
   .IF <STRLEN> = 0 .SETF BATCH
   .IF <STRLEN> = 0 .ASKS DEVICE Enter device name:
   .IF <STRLEN> = 0 .SETS DEVICE "DK:"

   .TEST DEVICE ":"
   .IF <STRLEN> = 0 .SETS DEVICE DEVICE+":"

   .SETS VOLID P2
   .TEST VOLID
   .IF <STRLEN> = 0 .SETF BATCH

   .SETS OWNER P3
   .TEST OWNER
   .IF <STRLEN> = 0 .SETF BATCH

.; Find out size of device, and whether it is online at present

   .TESTDEVICE 'DEVICE'
   .PARSE <EXSTRI> "," PHDEV size nl1 nl2 nl3 loaded online mount attach
   .IF SIZE = "" .SETS SIZE "0"
   .SETN DEVSIZ 'SIZE'
   .TEST ONLINE "onl"
   .IF <STRLEN> = 0 .GOTO ERROR1
```

```
.; Magnetic tapes represent a peculiarity in that their size is
.; undefined. We find out about it here, as it has an impact on
.; setting the directory segments.  You cannot parse "phdev" to
.; find MT, MS, MM, or MU.  Instead, IND returns "NSD" for tape.

   .SETF MAGTAP
   .IF DEVSIZ <> 0 .GOTO GETVOL

   .SETT MAGTAP

 ;Device 'DEVICE' returns a size of 'DEVSIZ'

   .ASK [MAGTAP] MAGTAP Is this a mag tape?  ['MAGTAP']
   .IFT MAGTAP  .GOTO GETVOL
   .ASKN [DEVSIZ] DEVSIZ What is the proper size for 'device' ?
   .IF DEVSIZ = 0 .GOTO ERROR2

.; The .VOL directive cannot be used with magtapes and
.; does not return the owner, if the volume is a magtape.

.GETVOL:
   .IFF MAGTAP .VOL OLDVOL 'DEVICE'
   .TEST VOLID
   .IF <STRLEN> > 0 .GOTO OWNER
   .IFT MAGTAP .GOTO NEWID
   ;
   ;The current volume id is "'OLDVOL'".
   ;(If this name is nonsense, the device has never been initialized).
   ;
.NEWID:
   .ASKS [0:12.] VOLID What is the new volume id ?
   .IFF <DEFAUL> .GOTO OWNER

.USEOLD:
   .SETF OLDOK
   .ASK [OLDOK] OLDOK Use old volume id ?
   .IFT OLDOK .SETS VOLID "'OLDVOL'"
   .IFF OLDOK .GOTO NEWID

.OWNER:
   .TEST OWNER
   .IF <STRLEN> > 0 .GOTO FILES
   .ASKS [1:12.:"'DEFOWN'"] owner Who "owns" volume 'VOLID' ?

.; If the user specified a maximum number of files, compute the
.; number of segments needed to hold it.  Will use the string
.; size of P4 to determine if the parameter is specified.

.FILES:
   .TEST P4
   .IF <STRLEN> = 0 .GOTO DUPDEF    ; No parameter, get defaults

   .SETN FILES 'P4'                 ; Retrieve desired number of files
   .GOSUB MAXFIL                    ; Determine space for files
   .IFT BATCH .GOTO INITDV          ; If batch processing, go initialize

.; When no maximum number of files is specified, determine a default
.; number of directory segments, using the DUP defaults.
```

```
.DUPDEF:
   .IFT MAGTAP .GOTO END2
   .SETN SEGS 0
   .SETN SEGN 1
.LOOP1:
   .IF DEVSIZ < 0  .GOTO END1
   .IF DEVSIZ > SEGSZ'SEGN' .GOTO END1
   .INC SEGN
   .GOTO LOOP1
.END1:
   .SETN SEGS SEGBL'SEGN'
   .SETN FILES (SEGFIL*SEGS)-EXTRA

.; If "batch processing", skip following interactive section

   .IFT BATCH .GOTO INITDV

.LOOP2:
   .GOSUB MAXFIL                    ; Determine space for files
   ;
   ;There is room for 'BLOCKS' blocks and 'FILES' files in the directory.
   .ASK [<FALSE>] CHANGE Change directory size ?
   .IFF CHANGE .GOTO END2
   .ASKN [::0] FILES How many files will you put on the device ?
   ;The directory now has 'SEGS' segments
   .GOTO LOOP2
.END2:

   ;
   ;I am about to initialize device "'DEVICE'", formerly called
   ;"'OLDVOL'", to "'VOLID'", with owner "'OWNER'".  This can cause
   ;loss of files!
   ;
   .ASK GODOIT Are you sure you want to do this ?
   .IFF GODOIT .GOTO SKIPIT

.INITDV:
   .OPEN INITXX.TMP
   .IFT MAGTAP .DATA INIT/VOL/NOQ 'DEVICE'
   .IFF MAGTAP .DATA INIT/VOL/NOQ/SEGMENT:'SEGS'. 'DEVICE'
   .DATA 'VOLID'
   .DATA 'OWNER'
   .CLOSE
   $@INITXX.TMP
   .IF <EXSTAT> <> <SUCCES> .GOTO ERROR3

   ;
   ;Device 'DEVICE' has been initialized successfully as volume 'VOLID'
   ;

   .GOSUB CLENUP                       ; Clean up before exit
   .EXIT

.; Alternative exit if user decides not to initialize at this time.

.SKIPIT:
   ;
   ;You chose not to initialize device 'DEVICE' at this time.
   ;
   .GOSUB CLENUP                       ; Clean up before exit
   .EXIT

.; SUBROUTINES

.; MAXFIL is entered with a number of files to go on the device.
.; It computes the minimum number of directory segments needed to
.; store that many files, the number of blocks available on the
.; device for file storage, and the maximum number of files
.; which will fit in that many directory segments.  In calculating
.; the number of segments, 7-byte directory entries are assumed,
.; and three free entries are left for the USR to shuffle.

.; VARIABLES
.; Files on input   - User-specified maximum number of files for device
.; Files on output  - Maximum number of files which fit directory size
.; Segs on output   - Number of directory segments needed to hold files
.; Blocks on output - Number of blocks available for file storage

.MAXFIL:
   .SETN SEGS (FILES+EXTRA+SEGFIL-1)/SEGFIL
   .IF SEGS > SEGBL1 .SETN SEGS SEGBL1
   .SETN FILES (SEGFIL*SEGS)-EXTRA
   .SETN BLOCKS DEVSIZ-((SEGS*2)+SYSBLK)
   .RETURN

.; CLENUP gets rid of any temporary files which were created

.CLENUP:
   .TESTFILE INITXX.TMP
   .IF <FILERR> = <SUCCES> $DELETE/NOQUERY INITXX.TMP
   .RETURN

.; Error exits

.ERROR1:
   ;
   ;NEWDEV - E - Device ’device’ not online or does not exist

   .GOSUB CLENUP                             ; Clean up and
   .EXIT <ERROR>

.ERROR2:
   ;
   ;NEWDEV - E - Check out device ’device’ and try again

   .GOSUB CLENUP
   .EXIT <ERROR>

.ERROR3:
   ;
   ;NEWDEV - E - Initialization of device ’device’ failed!
   ;
   ;Correct the problem (probably someone else has access)
   ;and try again
   ;

   .GOSUB CLENUP
   .EXIT <ERROR>

.; Debugging exit on errors.  Simply dumps the symbol table.
```

```
.DEBUG:
   .DUMP
   .EXIT <ERROR>
```

INIT.COM

```
.; TITLE:   INIT.COM
.; PURPOSE:
.;      Initializes a device, but only if it has not previously
.;      been initialized.  Provides a safe way of ensuring that
.;      VM: is initialized when the system is booted, without
.;      erasing VM's contents during a reboot.  This can be run
.;      from you system's STRTxx.COM file.

.; CALLING SEQUENCE:
.;        IND INIT.IND [DEV] [VOLID] [OWNER] [SWITCH]
.;    where the (optional) parameters are:
.;        DEV        Device (logical or physical) to be initialized
.;        VOLID        Volume identification string (no embedded spaces!)
.;        OWNER        Owner string (no embedded spaces!)
.;        SWITCH        Any additional switches for init
.;              (for example, /SEG:2)

.; PROCEDURE:
.; Initialization

   .ENABLE QUIET
   .ENABLE SUBSTITUTION
   .DISABLE SUFFIX
   .DISABLE PREFIX

   .SETS versn "INIT, Version 1"

   .SETS defvol "RT-11 Device"       ; Default device name
   .SETS defown "User"               ; default owner name

   ;
   ;'versn'
   ;

   .SETS DEVICE P1

   .TEST DEVICE
   .IF <STRLEN> = 0 .ASKS DEVICE Enter device name:
   .IF <STRLEN> = 0 .SETS DEVICE "dk:"

   .TEST DEVICE ":"
   .IF <STRLEN> = 0 .SETS DEVICE DEVICE+":"

.; See if it is possible to open a file on the device.  If it is,
.; the device exists with a directory, so exit immediately.

   $DIR/OUTPUT:NL: 'DEVICE'
   .IF <EXSTAT> = <SUCCES> .GOTO EXIT1

.; Cannot open a file on device, so try to initialize.  Allow user
.; to specify volume id and owner.  If no parameters are provided
.; in command line, use defaults defined above.

   ;INIT - I - disregard DIR error, will initialize 'device'
   .SETS VOLID P2
   .TEST VOLID
   .IF <STRLEN> = 0 .SETS VOLID "'DEFVOL'"

   .SETS OWNER P3
   .TEST OWNER
   .IF <STRLEN> = 0 .SETS OWNER "'DEFOWN'"

   .SETS FLAGS P4
   .TEST FLAGS
   .SETN LEN <STRLEN>
   .TEST FLAGS "/"
   .IF <STRLEN> = 1 .SETS FLAGS FLAGS[2:LEN]
   .TEST FLAGS
   .IF <STRLEN> > 0 .SETS FLAGS "VOL/NOQUERY/'FLAGS'"
   .IF <STRLEN> = 0 .SETS FLAGS "VOL/NOQUERY"

.; Now build the .COM file to initialize the device.
.; Note that file deletes itself.

   .OPEN INIT.TMP
   .DATA INIT/'FLAGS' 'DEVICE'
   .DATA 'VOLID'
   .DATA 'OWNER'
   .DATA DELETE INIT.TMP
   .CLOSE

.; Finally, execute it

   $@INIT.TMP

   .IF <EXSTAT> = <SUCCES> .GOTO EXIT2
   .GOTO EXIT3                       ; Failure


.; Exits

.; Exit 1 -- device is already initialized

.EXIT1:
   ;INIT - I - Device 'device' is already initialized.
   .EXIT <SUCCES>

.; Exit 2 -- device successfully initialized

.EXIT2:
   ;INIT - I - Device 'device' successfully initialized.
   .EXIT <SUCCES>


.; Exit 3 -- something about the initialization failed.

.EXIT3:
   ;INIT - W - Initialization of 'device' failed.
   .EXIT <WARNIN>
```

GENCMD.COM

```
.; TITLE:  GENCMD.COM
.; REQUIREMENTS: {KED|KEX} as system default editor, IND and DIR
.; PURPOSE: Generates command files.
.;          Uses DK:GenCmd.Tmp as temporary file name
.;
.; PROCEDURE:
   .DISABLE PREFIX, SUFFIX
   .ENABLE SUBSTITUTION,ESCAPE
   .GOTO ASK
   .HELP:
   ;
   ; This procedure generates a command file containing a line
   ; for each file matching the selection criteria (any
   ; directory selection string).  The command lines are based
   ; on the command prototype specified, with the file name
   ; substituted for each "^" in the prototype.
   ;
   ;EXAMPLE:
   ;
   ;Select (*.*)? *.MAC/New
   ;Command ($Type ^)? $Copy ^ NL:
   ;Output File (TEMP.COM)? BBUCKT.COM
   ;
   ;If the files FOO.MAC and BAR.MAC exist, it generates
   ;a file called BBUCKT.COM which contains:
   ;
   ;$COPY FOO.MAC NL:
   ;$COPY BAR.MAC NL:
   ;
.; Collect input parameters
   .ASK:
   .SETS PPatte "*.*"
   .ASKS [0::PPatte] Patter Select ('PPatte')?
   .IFT <Escape> .GOTO HELP
   .SETS PCmnd "$Type ^"
   .ASKS [0::PCmnd] Cmnd Command ('PCmnd')?
   .IFT <Escape> .GOTO Help
   .SETS POut "Temp.Com"
   .ASKS [0::POut] Out Output File ('POut')?
   .IFT <Escape> .GOTO HELP
.; Get the file names before we create temporary files
   $DIRECTORY 'Patter'/BRIEF/COLUMN:1/OUT:'Out'
.; Define KED keystrokes
   .SETN EscVal 27.
   .SETS Esc "'EscVal%V'"
   .SETS APP "'Esc'O"
   .SETS CSI "'Esc'["
   .SETS Enter "'APP'M"
   .SETS Gold "'APP'P"
   .SETS Substi "'Gold''APP'M"
   .SETS Find "'Gold''APP'R"
   .SETS FindNe "'APP'R"
   .SETS Execut "'Gold'x"
   .SETS Select "'APP'n"
   .SETS StopDe "'Gold's"
   .SETS Up "'CSI'A"
   .SETS Left "'CSI'D"
   .SETS Top "'Gold''Up'"
   .SETS Bottom "'Gold''CSI'B"
   .SETS DelCha "'APP'l"
   .SETS BLine "'APP'p"
   .SETS DelLin "'APP'S"
   .SETS Word "'APP'q"
   .SETS Cut "'APP'v"
   .SETS Paste "'Gold''Cut'"
   .SETS KComma "'Gold''APP'w"

.; Count the "^"s in the prototype command line
   .SETS TempS Cmnd
   .SETN Rpt 0
   .LOOP:
       .TEST TempS
       .SETN TempN <StrLen>
       .TEST TempS "^"
       .IF <StrLen> = 0 .GoTo LoopX
       .INC Rpt
       .SETS TempS TempS[<StrLen>+1:TempN]
       .GOTO LOOP
   .LOOPX:
.; Generate the KED macro definition to build requested command file
   .OPEN GenCmd.Tmp
.; Delete top line and bottom two lines
   .DATA 'Enter''DelLin''Bottom''Up''Up''DelLin''DelLin''Top''KComma'
.; Compress spaces out of file names
   .DATA 'Enter''Find' 'FindNe''KComma'
   .DATA Learn'Enter''Substi''StopDe''Execut''KComma'
   ;        17856=72*31*8 (72 files/seg, 31 seg/disk, 8 max sp/name)
   .DATA 'Enter''Gold'17856'Execut''Top''KComma'Clear Macro \
.; Generate a command line for each file found
   .DATA Set Search End'Enter''Find'^'Enter''KComma'Learn'Enter''Cmnd'
   .DATA 'Select''Word''Cut''Up''KComma'
.; Substitute the file name for each "^" in the prototype command line
   .LOOP1:
       .DEC Rpt
       .IF Rpt = -1 .GoTo LupX
       .DATA 'Enter''FindNe''Left''DelCha''Paste''KComma'
       .GOTO LOOP1
   .LUPX:
   .DATA 'Enter''BLine''DelLin''StopDe''Kcomma'
   ;        2232=72*31 (72 files/seg, 31 seg/disk)
   .DATA 'Enter''Gold'2232'Execut''Up''DelLin''KComma'Exit
   .CLOSE
.; Transform .DIR to .COM, and delete temp file
   $EDIT 'Out'/RECOVER:GenCmd.Tmp
   $DELETE GenCmd.Tmp
   .DISABLE SUBSTITUTION
   .EXIT
```

Index