ENABLE™ Reference Manual

Abstract This manual describes the operation of ENABLE and the syntax of the ENABLE commands. This manual is intended for all users of ENABLE.

Part Number 82560

Tandem Computers Incorporated

Document History

Edition	Part Number	Operating System Version	Date
First Edition	82560 A00	GUARDIAN B00	March 1985

New editions incorporate any updates issued since the previous edition.

CONTENTS

FIGURES

TABLES

PREFACE

This manual describes the syntax of ENABLE, a product that is part of the ENCOMPASS distributed data base management system. ENABLE allows you to build simple applications that execute within a PATHWAY system.

The following manuals contain more detailed information about the Tandem NonStop II and TXP computer systems and the software products used with ENABLE:

- Data Definition Language (DDL) Reference Manual
- ENABLE User's Guide
- ENFORM Reference Manual
- GUARDIAN Operating System User's Guide
- GUARDIAN Operating System Programmer's Guide
- PATHWAY System Management Reference Manual
- PATHWAY SCREEN COBOL Reference Manual

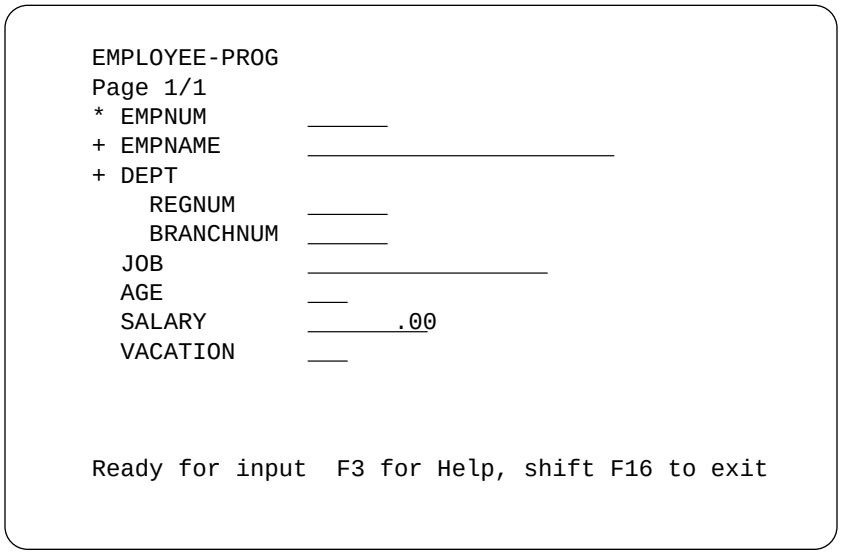
SYNTAX CONVENTIONS IN THIS MANUAL

The following list summarizes the conventions for syntax notation in this manual.

Notation	Meaning
UPPERCASE LETTERS	Uppercase letters represent keywords and reserved words; you must enter these items exactly as shown.
<lowercase letters>	Lowercase letters within angle brackets represent variables that you must supply.
Brackets []	Brackets enclose optional syntax items. A vertically aligned group of items enclosed in brackets represents a list of selections from which you may choose one or none.
Braces {}	Braces enclose required syntax items. A vertically aligned group of items enclosed in braces represents a list of selections from which you must choose only one.
Ellipsis ...	An ellipsis immediately following a pair of brackets or braces indicates that you can repeat the enclosed syntax items any number of times.
Percent Sign %	Precedes a number in octal notation.
I/O	In procedure calls, input parameters (those that pass data from the calling program to the called procedure) are followed by an 'I' (input). Output parameters (those that return data from the called procedure to the calling program) are followed by an 'O' (output).
Spaces	If two items are separated by a space, that space is required between the items. If one of the items is a punctuation symbol, such as a parenthesis or a comma, spaces are optional.
Punctuation	Parentheses, commas, semicolons, and other symbols or punctuation not described above must be entered precisely as shown. If any of the punctuation above appears enclosed in quotation marks, that character is not a syntax descriptor but a required character and you must enter it as shown.
RETURN	Indicates a carriage return.

GENERATING AN IMMEDIATE SINGLE-FILE APPLICATION

Enter the commands exactly as shown to create a data base file and an application that can be used to enter employee data.



(1) CREATE A DATA DICTIONARY

```
:EDIT ddlsrc1

*ADD
   1   RECORD EMPLOYEE.
   2   FILE IS empfile              KEY-SEQUENCED.
   3   02 EMPNUM                    PIC 9(4).
   4   02 EMPNAME                   PIC X(18).
   5   02 DEPT.
   6       04 REGNUM                PIC 9(4).
   7       04 BRANCHNUM             PIC 9(4).
   8   02 JOB                       PIC X(12).
   9   02 AGE                       PIC 99.
  10   02 SALARY                    PIC 9999V99.
  11   02 VACATION                  PIC S99.
  12   KEY IS EMPNUM.
  13   KEY "EN" IS EMPNAME.
  14   KEY "DP" IS DEPT.
  15   END
  16   / /
*EXIT

:DDL/IN ddlsrc1/DICT, FUP fupsrc1 !
```

(2) CREATE A DATA BASE FILE

```
:FUP/IN fupsrc1/
```

(3) CALL ENABLE AND GENERATE AN APPLICATION

```
:ENABLE

%SET BOX RECORD EMPLOYEE
%ADD BOX EMPLOYEE
%SET APPL TREE (01 EMPLOYEE)
%SET APPL PATHCOMFILE singlpth !
%ADD APPL EMPLOYEE-PROG
%GENERATE EMPLOYEE-PROG
%EXIT
```

(4) ESTABLISH A PATHWAY SYSTEM AND EXECUTE THE APPLICATION

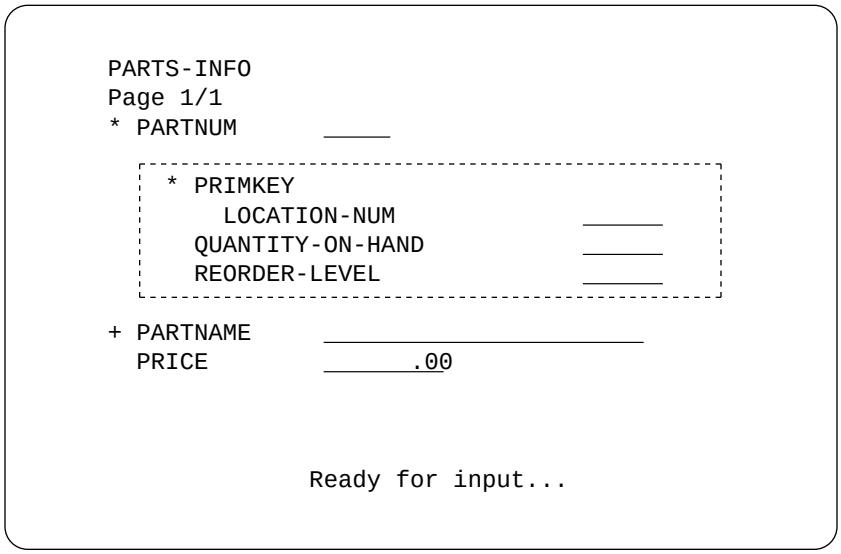
```
:EDIT enabex1

*ADD
   1   PURGE enablog, enabctl
   2   CREATE enablog
   3   ASSIGN PATHCTL, enabctl
   4   PATHMON/NAME $one,CPU 0, NOWAIT,OUT enablog/
   5   PATHCOM/IN singlpth/$pm
   6   PATHCOM $one;RUN EMPLOYEE-PROG
   7   PATHCOM $one;shutdown,wait
   8   / /
*EXIT

:OBEY enabex1
```

GENERATING AN IMMEDIATE MULTIPLE-FILE APPLICATION

Enter the following commands exactly as shown to create two data base files and an application that can be used to enter data in the files.



(1) CREATE A DATA DICTIONARY

```
:EDIT ddlsrc2
*ADD
    1   RECORD PARTS.
    2   FILE IS parts              KEY-SEQUENCED.
    3   02 PARTNUM                 PIC 9(4) HEADING “PART NUMBER”.
    4   02 PARTNAME                PIC X(20).
    5   02 PRICE                   PIC 999999V99.
    6   KEY 0 IS PARTNUM
    7   KEY “pn” IS PARTNAME.
    8   END
    9
   10   RECORD INVENTORY.
   11   FILE IS inventry           KEY-SEQUENCED
   12   02 PRIMKEY.
   13      04 PARTNO               PIC 9(4)
   14      04 LOCATION             PIC XXX HEADING “LOCATION NO.”.
   15   02 QUANTITY-ON-HAND        PIC 99999.
   16   02 REORDER-LEVEL           PIC X(5)
   17   KEY 0 IS PRIMKEY.
   18   END
   19   / /
*EXIT

:DDL/IN ddlsrc2/DICT, FUP fupsrc2!
```

(2) CREATE THE DATA BASE FILES

```
:FUP/IN fupsrc2/
```

(3) GENERATE THE APPLICATION

```
:EDIT enabsrc1
 ADD
    1   SET BOX RECORD PARTS
    2   ADD BOX PARTS
    3   SET BOX RECORD INVENTORY
    4   ADD BOX INVENTORY
    5   SET APPL TREE (01 PARTS
    6     02 INVENTORY
    7       LINK PARTS.PARTNUM TO OPTIONAL INVENTORY.PARTNO)
    8   SET APPL PATHCOMFILE multipth !
    9   ADD APPL PARTS-INFO
   10   GENERATE APPL PARTS-INFO
   11   / /
*EXIT

:ENABLE/IN enabsrc1/
```

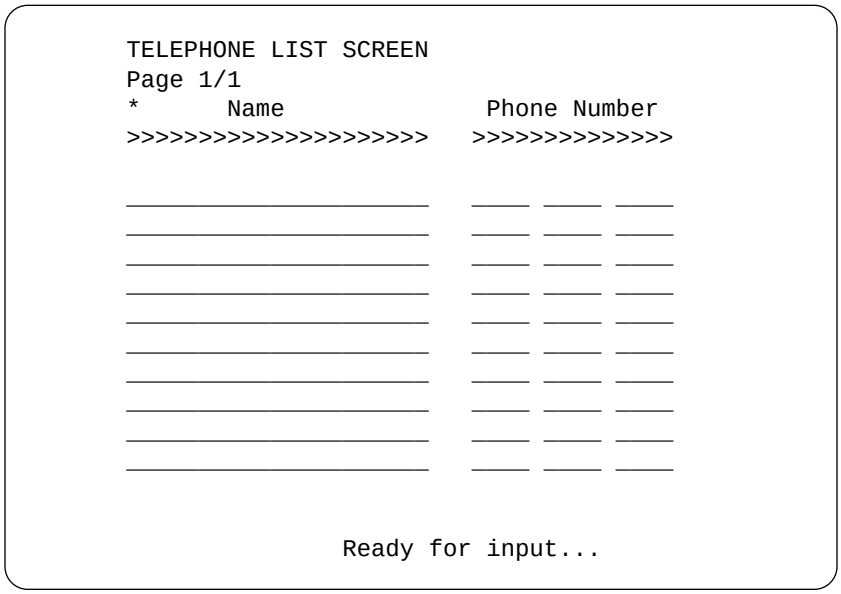
(4) ESTABLISH A PATHWAY SYSTEM AND EXECUTE THE APPLICATION

```
:EDIT enabex2
 *ADD
    1   PURGE multilog, multictl
    2   CREATE multilog
    3   ASSIGN PATHCTL, multictl
    4   PATHMON/NAME $mult, CPU 0, NOWAIT, OUT multilog/
    5   PATHCOM/IN multipth/$mult
    6   PATHCOM $mult;RUN PARTS-INFO
    7   PATHCOM $mult;SHUTDOWN,WAIT
    8   / /
*EXIT

:OBEY enabex2
```

GENERATING AN APPLICATION WITH A TABULAR SCREEN

Enter the commands as shown to create a telephone book file and generate an application that displays the following screen:



(1) CREATE A RECORD DESCRIPTION

```
:EDIT ddlsrc3
*ADD
    1   RECORD PHONE-BOOK.
    2   FILE IS phones              KEY-SEQUENCED.
    3   02 NAME                     PIC X(20)
    4   02 PHONE-NUMBER.
    5       04 AREA-CODE            PIC 9(3)
    6       04 PREFIX               PIC 9(3)
    7       04 NUMBER               PIC 9(4)
    8   KEY 0 IS NAME.
    9   END
   10   / /
*EXIT

:DDL/IN ddlsrc3/DICT, FUP telefup
```

(2) CREATE THE FILE

```
:FUP/IN telefup/
```

3) GENERATE THE APPLICATION

```
:EDIT enabsrc3
*ADD
    1   SET RECORD PHONE-BOOK
    2   SET SCREENFORMAT COMPRESSED
    3   SET HEADINGS NULL
    4   SET BOXTITLE 1 “              Name                Phone Number”
    5   SET BOXTITLE 2 “ >>>>>>>>>>>>>>>>>>>>>>     >>>>>>>>>>>>>>“
    6   SET BOXTITLE 3 “    “
    7   SET SIZE 10
    8   ADD BOX PHONE-BOX
    9   SET APPL TREE (01 PHONE-BOX)
   10   SET APPL PATHCOMFILE telepath !
   11   SET APPL TITLE “TELEPHONE LIST SCREEN”
   12   ADD APPL TELE-PROG
   13   GENERATE APPL TELE-PROG
   14   / /
*EXIT

:ENABLE/IN enabsrc3/
```

(4) ESTABLISH A PATHWAY SYSTEM AND EXECUTE THE APPLICATION

```
:EDIT enabobey
*ADD
    1   PURGE log3, enabctl3
    2   CREATE log3
    3   ASSIGN PATHCTL, enabctl3
    4   PATHMON/NAME $TWO, CPU 0, NOWAIT, OUT LOG3/
    5   PATHCOM/IN TELEPATH/$TWO
    6   PATHCOM $TWO;RUN TELE-PROG
    7   PATHCOM $TWO;SHUTDOWN,WAIT
    8   / /
*EXIT

:OBEY enabobey
```

SECTION 1

INTRODUCTION

ENABLE builds an interactive update application that executes under the PATHWAY transaction processing system. Like other applications defined and developed for the PATHWAY environment, an ENABLE application implements transactions that originate at a terminal and access a data base.

ENABLE generates a simple interactive application under PATHWAY by performing the following tasks:

- Producing SCREEN COBOL source code that defines and controls terminal display screens
- Passing the source code to the SCREEN COBOL compiler for compilation
- Providing a server program to support file I/O for the application
- Providing a command file that you use to create a PATHWAY configuration for execution of the application

An application generated by ENABLE provides you with immediate access to one or more files under PATHWAY without requiring any custom programming. You can use the application for temporary or simple requirements, as a base for application development, or as a module in a more extensive PATHWAY application.

This manual describes the syntax of the ENABLE commands. You can use these commands to generate applications that can access a single data base file or applications that can access several data base files.

For a description of the individual tasks necessary to create an application, refer to the ENABLE User's Guide.

APPLICATIONS GENERATED BY ENABLE

Figure 1-1 shows the screen displayed by a sample ENABLE application. You could use this application to retrieve and update the information in a single data base file. Because applications similar to this sample application can access a single data base file, these types of applications are sometimes called single-file applications.

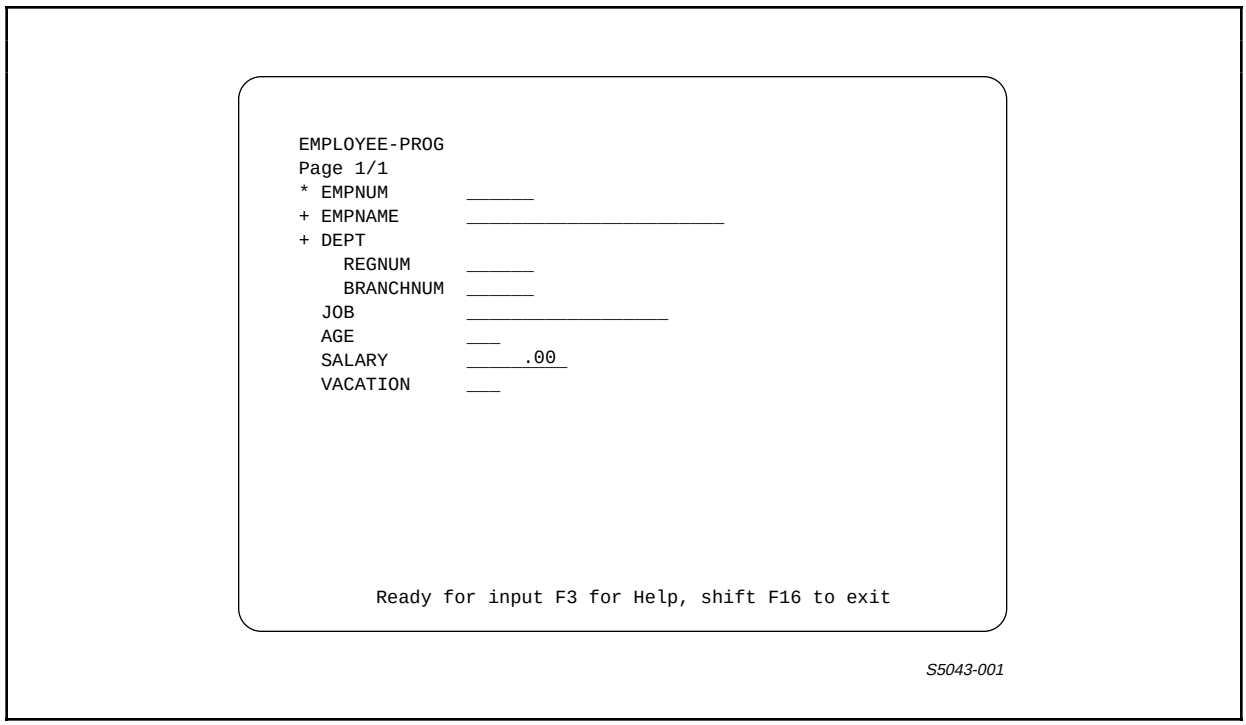


Figure 1-1. Sample Single-File Application Display Screen

When you generate a single-file application, ENABLE allows you to:

- Define the operations (delete, insert, read, and update) that the application can perform on the file
- Control the format of the screen displayed by the application

- Define the method used to maintain the integrity of the file

To generate a single-file application, you use the ENABLE commands described in Section 3.

Figure 1-2 shows the screen displayed by a sample application that you could use to retrieve or update records in two data base files. Because applications similar to this sample application can access two or more files, these types of applications are sometimes called multifile applications.

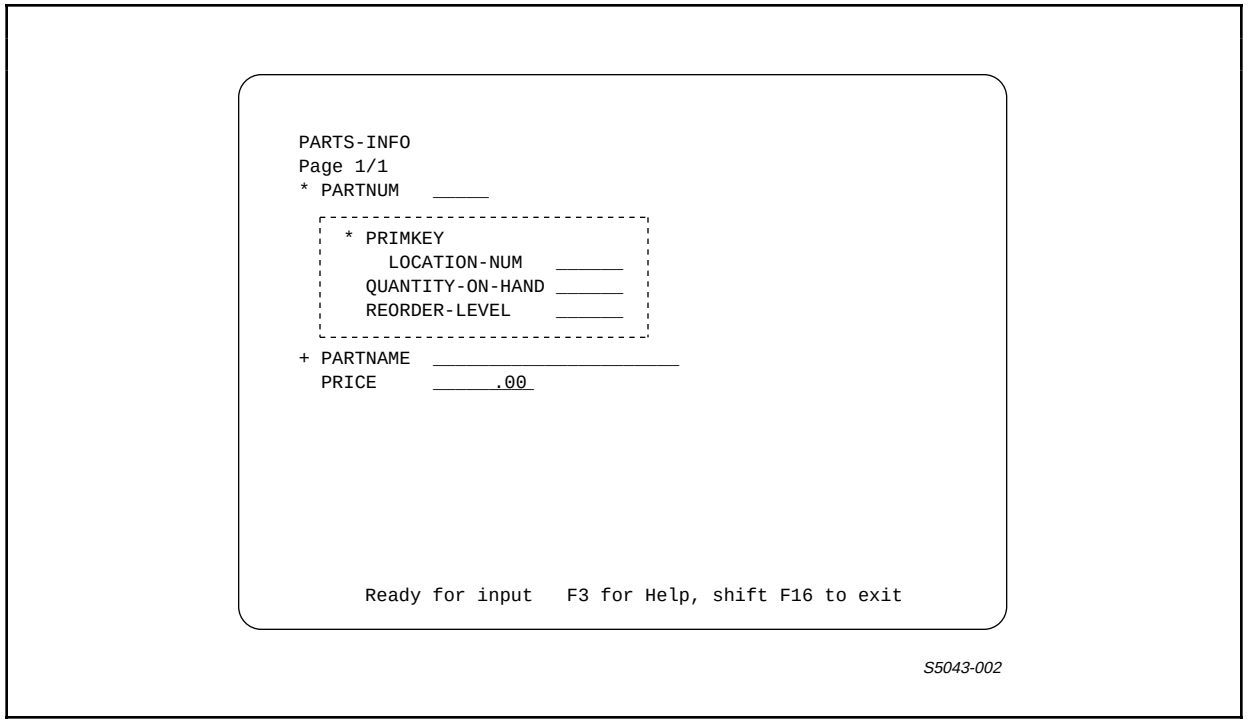


Figure 1-2. Sample Multifile Application Display Screen

When you generate a multifile application, ENABLE allows you to:

- Define the operations (delete, insert, read, or update) that the application can perform on each file
- Control the format in which information from each file is displayed on the screen
- Determine the method used to ensure the integrity of each file

You generate a multifile application by using the ENABLE commands described in Section 3.

ENABLE COMPONENTS

To generate an application, ENABLE uses the following components: the ENABLE compiler, the program generator, and a General Server. ENABLE provides two skeleton files: a SCREEN COBOL application program skeleton and a PATHCOM command file skeleton.

ENABLE uses the data dictionary that is produced when the Data Definition Language (DDL) compiler creates the description of a data base. The dictionary provides record, field, and key information.

ENABLE Compiler

The ENABLE compiler, ENABLE, runs under the command interpreter. The compiler performs the following functions:

- Examines the syntax of ENABLE commands for errors
- Opens the dictionary and obtains information about the structure of each file to be used by an application
- Stores information about applications to be generated in a special table called the object table
- Passes the object table to the Program Generator when you indicate you want to generate an application

Program Generator

The program generator, ENABLOBJ, runs at the request of the ENABLE compiler; it performs the following functions:

- Uses the information in the object table to transform the SCREEN COBOL skeleton file into SCREEN COBOL source code
- Calls the SCREEN COBOL compiler, which compiles the object code to a default or designated object file

- Uses the information in the object table to transform the PATHCOM skeleton file into a PATHCOM command file

General Server

The General Server, ENABLEGS, accesses and updates the data base files. ENABLEGS can access up to 32 data base files. Although ENABLEGS is an integral part of the ENABLE environment, it is available for use as a server with other applications. Details regarding individualized use of ENABLEGS appear in Appendix E.

Application Program Skeleton File

The application program skeleton file, ENABAPPS, provides the basis for the SCREEN COBOL source code generated by the program generator. An application programmer can customize the program skeleton and use that customized version in the generation process. Appendix D provides general guidelines for customizing the program skeleton.

PATHCOM Command Skeleton File

The PATHCOM command skeleton file, ENABPATS, provides the basis for the PATHCOM command file generated by the program generator. You can use this command file to configure a PATHWAY system for execution of the application.

The PATHCOM command file generated by the program generator selects defaults to ensure that the required PATHWAY parameters are present. The generated PATHCOM command file causes a PATHWAY cold start, that is, a start in which no previous PATHWAY configuration exists.

If the application generated by ENABLE is to be integrated into an existing PATHWAY system, the existing PATHCOM command file must be customized. The generated command file can be used as a starting point.

Dictionary Files

ENABLE accesses DDL data dictionary files to extract information about the data base files to be used by an application. The dictionary must contain a record description of each file to be used. The record description must be compatible with SCREEN COBOL representation capabilities. Specifically, this means that the DDL statements and clauses used to create the record description must conform to the following rules:

- A TYPE clause must be CHARACTER or BINARY.
- OCCURS clause nesting must not exceed four levels; the OCCURS value (number of occurrences) must not exceed 999.
- Elementary items must not exceed 256 bytes.
- The record must not exceed 2046 bytes. For key-sequenced files, the record cannot exceed 2035 bytes.
- FILLER and REDEFINES items can appear in a record description; however, these items do not appear on the screen. The application displays only named items and the first declaration of a redefined structure.
- When several fields constitute a record key and have the same byte offset, only the field at the highest level of the group is available as a key.
- When an OCCURS DEPENDING ON clause appears, ENABLE uses the maximum possible record size. If an application reads a record that is shorter than the maximum length, the General Server pads the records with blanks. This padding can cause some problems in numeric fields.

  The application uses the maximum record size when writing all records with OCCURS DEPENDING ON clauses to the data base. Therefore, if you update a variable-length record that is shorter than the maximum size, the application writes out a record that is of the maximum size and ignores the value of the item upon which the OCCURS depends. For entry-sequenced files, such an update results in an error.
- BINARY 8, FLOAT 32, FLOAT 64, COMPLEX, LOGICAL 2 and LOGICAL 4 data items are not supported by ENABLE.
- If a HEADING clause appears, ENABLE uses only the first 28 characters of the heading.

- If a VALUE clause appears, ENABLE accepts an initial value of up to 30 characters in length. Longer values are discarded.

OVERVIEW OF APPLICATION GENERATION

You can generate an ENABLE application by supplying ENABLE with:

1. A data dictionary that describes the files to be accessed by your application

2. ENABLE commands that define the specifications of your application

Figure 1-3 provides an overview of application generation by comparing the elements that you supply with the elements supplied by ENABLE.

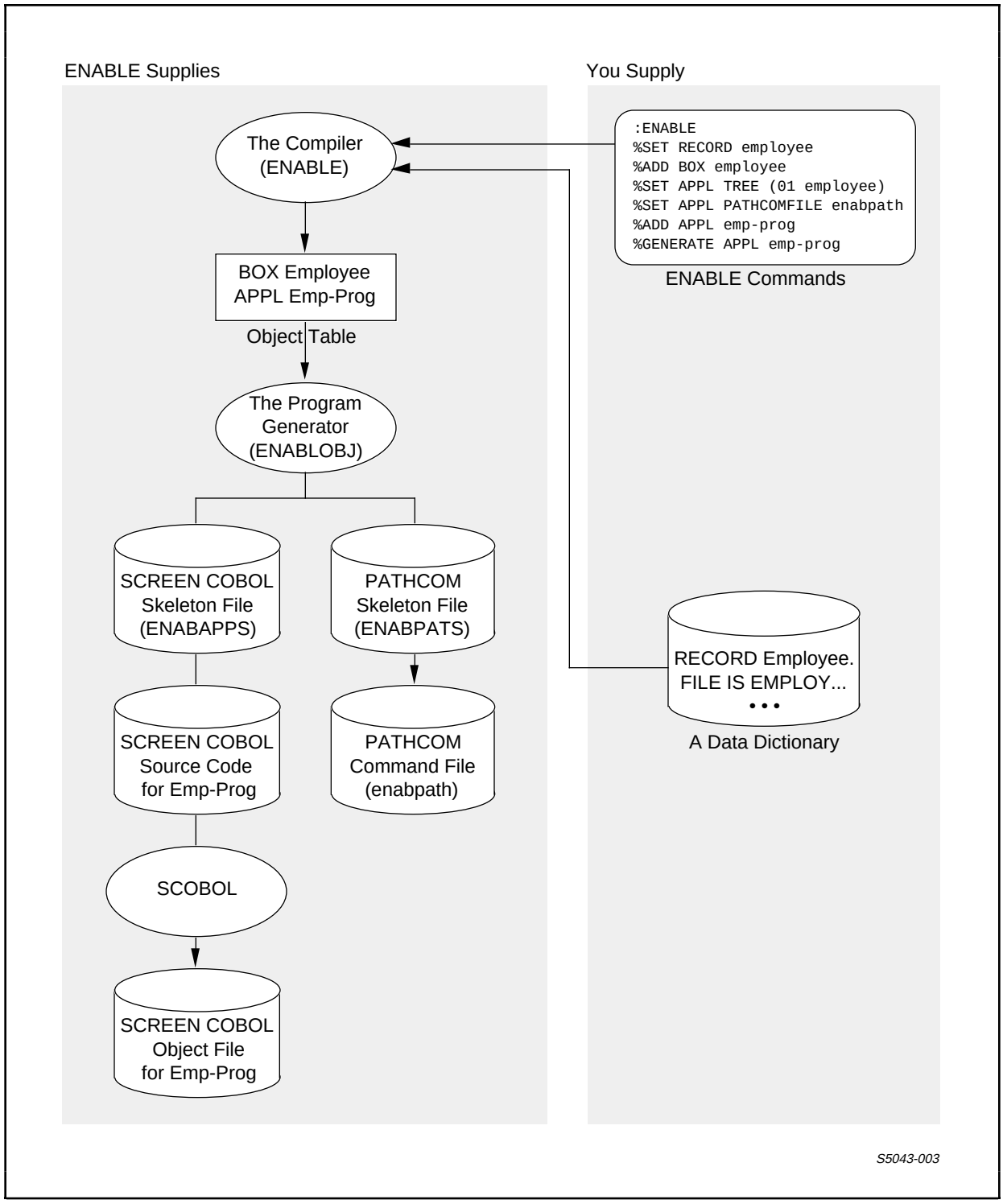


Figure 1-3. Overview of Application Generation

APPLICATION EXECUTION

To execute an application generated by ENABLE, you must:

1. Supply the data base files to be accessed by the application

2. Establish a PATHWAY system to execute the application

Figure 1-4 illustrates the tasks that you perform to establish a PATHWAY system and execute an application.

To establish a PATHWAY system and execute an application, you enter a series of commands described in Section 5. These commands:

- Create and name a PATHWAY monitor process (PATHMON). PATHMON controls the interrelations of the processes and devices within the PATHWAY system. PATHMON maintains information about the environment of the PATHWAY system in a file called PATHCTL. PATHMON can report errors and changes in status to a log file.

- Create a PATHCOM process. PATHCOM, the command interface to PATHMON, passes the information in the PATHCOM command file generated by ENABLE to PATHMON. PATHMON stores this information in the PATHCTL file.

- Tell PATHCOM to execute the application. PATHCOM passes this information to PATHMON. PATHMON starts a terminal control process (TCP) using the characteristics described in the generated PATHCOM command file. The TCP performs application operations according to the SCREEN COBOL program, provides general control of the terminal, and sends messages to the General Server.

When you execute the application, the General Server, which is provided by ENABLE, accesses and updates the data base and replies to the TCP.

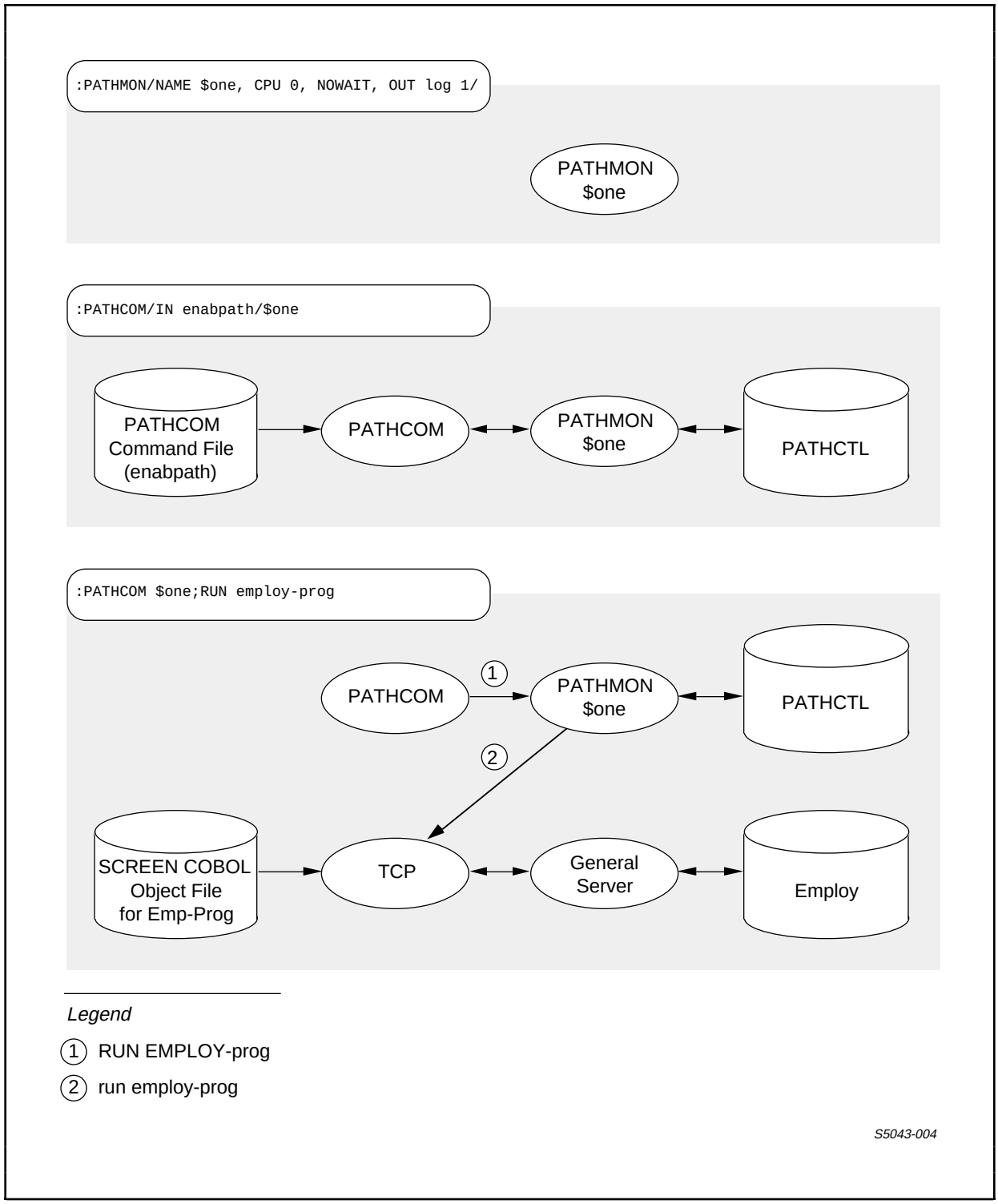


Figure 1-4. Overview of Application Execution

SUMMARY OF TASKS REQUIRED TO GENERATE AND EXECUTE AN APPLICATION

To generate and execute an ENABLE application, you must:

1. Use DDL to create the appropriate record descriptions and to compile them into a dictionary. DDL can also create File Utility Program (FUP) file creation source code that can be used in step 2.

2. Use FUP to create the physical files if the files do not already exist.

3. Call ENABLE and issue appropriate commands to generate the application.

4. Initiate PATHMON and use the PATHCOM command file generated during step 3 to establish the PATHWAY system.

If a data base and data dictionary already exist, only steps 3 and 4 need to be completed. Refer to the ENABLE User's Guide for guidelines to follow when you perform these tasks.

SECTION 2

RUNNING ENABLE

You start ENABLE by entering the ENABLE command in response to the command interpreter prompt. The syntax of the ENABLE command is:

```
ENABLE [ / <parameter> [ , <parameter> ] ... / ]
         [ <command> ] [ ; <command> ] ...

<parameter> is:

  IN <command-file>

     specifies a file from which commands are to be read.
     If you omit this option, ENABLE takes input from the
     current input file of the command interpreter,
     typically the terminal.

  OUT <list-file>

    specifies a file to which the listing is to be written.
    If you omit the OUT option, ENABLE directs output to the
    current output file of the command interpreter,
    typically the terminal.  Specifying OUT on the ENABLE
    command does not prohibit later use of the ENABLE
    operating command OUT.
```

--->**>

NAME $<process-name>

identifies a symbolic process name to be assigned to the new ENABLE process.

CPU <cpu-number>

identifies a processor where the ENABLE process will run.

PRI <priority>

specifies execution priority of the ENABLE components.

MEM <num-pages>

defines the maximum number of data pages allocated to ENABLE.

NOWAIT

returns control to the command interpreter while ENABLE executes.

<command> [; <command>]...

is one or more ENABLE commands separated by semicolons. When commands are present, ENABLE executes the commands and terminates without reading any command file.

For more information about these <parameters>, refer to the GUARDIAN Operating System User's Guide.

The following applies to the OUT option of the ENABLE command:

- If the OUT file specification is a disc file name and the file already exists, ENABLE appends the output listing to the end of the file. If the OUT file specification is a disc file name and the file does not exist, ENABLE creates an edit-type file for the output listing.
- If the OUT file specification is a line printer or a process, ENABLE indicates that a page eject is to be performed when the file is opened. ENABLE does not indicate that a page eject is to be performed before the file is closed.

- If the OUT file specification is a magnetic tape, ENABLE specifies that two consecutive file marks are written just before the file is closed.

INTERACTIVE MODE

ENABLE functions in interactive mode when you enter commands from a terminal keyboard. ENABLE prompts for a command by displaying the percent symbol (%); for example:

```
:ENABLE
ENABLE - T9155B00 - (01APR84)       SYSTEM \XYZ
%
```

When you enter a command, ENABLE executes the command and issues another prompt.

If you press the terminal BREAK key when you are using ENABLE in interactive mode, the following occurs:

- Pressing the BREAK key within a single or multiline ENABLE command causes ENABLE to ignore the command or command lines. ENABLE displays a prompt, and you can enter another command.

- Pressing the BREAK key when you are using the operating command FC (explained later in this section and in Section 4), causes ENABLE to ignore the FC command and the command line. ENABLE displays a prompt, and you can enter another command.

- Pressing the BREAK key at any other time, including on the ENABLE prompt, returns you to the command interpreter prompt without terminating ENABLE. If you type the word STOP in response to the command interpreter prompt, you terminate the ENABLE process. You do not terminate other processes, such as ENABLOBJ or SCOBOLX. Since these processes could still be running, you should type the command STATUS *,TERM to determine the CPU and process numbers (PIN) of any running processes. If you want to terminate these processes, type:

  STOP <cpu>, <pin>

NONINTERACTIVE MODE

ENABLE functions in noninteractive mode when you enter commands through a command file; for example:

```
:ENABLE /IN enabcomm, OUT $s/
```

In this example, ENABLE reads commands from an edit-type file named "enabcomm." When ENABLE encounters an end-of-file or an EXIT command in the command file, ENABLE terminates and control returns to the command interpreter.

ALLOCATING EXTENDED MEMORY FOR ENABLE

ENABLE uses an extended memory segment called the object table during application generation. The object table is an internal table that stores information about the applications to be generated. The GUARDIAN operating system allocates the extended memory for the object table when it starts ENABLE for you. By default, ENABLE requests 500 pages of extended memory for the object table.

Both the ENABLE compiler and the program generator share the object table. Either the compiler or the program generator can run out of space in the object table. When the object table is nearly full, ENABLE issues a warning message stating that it is low on extended memory. If you receive this warning message, you can use the SHOW, INFO, and OUT commands (explained later in this manual) to obtain information about the contents of the object table. You cannot, however, increase the amount of extended memory allocated without terminating and restarting ENABLE.

Before you enter the ENABLE run command, you can define the number of extended memory pages that are to be allocated by initializing a value for the EXTPAGES parameter. You initialize a value for this parameter by using the command interpreter PARAM command.

The syntax of the PARAM command is:

```
PARAM EXTPAGES <num-pages>

<num-pages>

  is the number of extended memory pages that you want to
  allocate.  <num-pages> must be an integer in the range of
  1 to 32765.  The default is 500 pages.
```

If the GUARDIAN operating system cannot allocate the specified or default number of extended memory pages, ENABLE issues an error message. In this case, you can use the EXTPAGES parameter to reduce the number of extended memory pages to be allocated.

COMMANDS

Two categories of commands exist. These categories are:

- Operating commands, which control ENABLE and its environment
- ENABLE commands, which are associated with the control and definition of objects

Operating Commands--Functional Overview

Operating commands control the environment of ENABLE. Section 5 describes the operating commands and their functions. Table 2-1 summarizes these commands.

Table 2-1. Operating Command Summary (Continued next page)

Command	Description
CMDSYS	Sets the default system name for expansion of any file names except obey file names. This command overrides a preceding SYSTEM command for all file names except obey file names.
CMDVOL	Sets the default volume and subvolume names for expansion of any file names except obey file names. This command overrides a preceding VOLUME command for all file names except obey file names.
ENV	Displays the current settings for the program environment parameters.
EXIT	Stops ENABLE.
FC	Displays the previous command line for you to edit or repeat.
HELP	Displays the syntax of ENABLE commands.
OBEY	Causes commands to be read from a specified file.
OBEYSYS	Sets the default system name for expansion of obey file names. This command overrides preceding SYSTEM command for obey file names only.
OBEYVOL	Sets the default volume and subvolume name for expansion of obey file names. This command overrides a preceding VOLUME command for obey file names only.
OUT	Directs the output listing to the specified file.

Table 2-1. Operating Command Summary (Continued)

Command	Description
SYSTEM	Sets the default system name for expansion of any file names including CMDSYS and OBEYSYS names. This command overrides a preceding CMDSYS or OBEYSYS command.
VOLUME	Sets the default volume and subvolume for expansion of any file names including CMDVOL and OBEYVOL names. This command overrides a preceding CMDVOL or OBEYVOL command.

ENABLE Commands--Functional Overview

ENABLE commands allow you to describe and define two types of objects: applications and boxes.

An application represents the SCREEN COBOL requester program, the associated PATHCOM command file, and the ENABLE supplied server program. You describe an application by supplying values for application attributes. The values of application attributes describe global characteristics of an application such as:

- The name of a PATHCOM command file
- The type of terminal upon which the SCREEN COBOL requester program runs

All of the ENABLE commands that apply to applications contain an optional keyword, APPL. When you start ENABLE, the default object type of all commands is APPL. This means that you do not have to include the keyword APPL when you enter a command that applies to an application. Be aware that you can change the default object type with one of the ENABLE commands.

A box represents a particular data base file to be accessed by an application. (The term box derives from the box that surrounds file information on the screen displayed by a multifile application.) You describe a box by supplying values for box attributes. The values of these attributes identify the characteristics of a data base file such as:

- The name of the record description that describes the file
- The operations (delete, insert, read, or update) that the application can perform on the file

All of the ENABLE commands that apply to boxes contain an optional keyword, BOX. When you start ENABLE the default object type for all commands is APPL. When you enter most commands that apply to a box, you can either include the keyword BOX or change the default object type (with the ASSUME command). To maintain compatibility with a previous version of ENABLE, you can describe a box even if the object type is APPL.

Table 2-2 summarizes the ENABLE commands and their functions.

Table 2-2. ENABLE Command Summary

Command	Description
ADD	Names an object and adds the object's description to the object table.
ASSUME	Establishes a default object type for subsequent ENABLE commands.
DELETE	Deletes a named object from the object table.
GENERATE	Generates an application.
INFO	Displays the attributes of a named object.
RESET	Resets attribute values.
SET	Changes the current value of an attribute.
SHOW	Displays the current values of attributes.

The function of several of the ENABLE commands is to supply values for application or box attributes. Table 2-3 summarizes the application attributes. Table 2-4 summarizes the box attributes.

Table 2-3. Functional Summary of Application Attributes

Attribute	Function
PATHCOMFILE	Identifies the name of the PATHCOM command file.
PATHCOMSKELETON	Identifies the name of the PATHCOM skeleton file.
SCOBOLCOMPILER	Identifies a SCREEN COBOL compiler to be used to produce SCREEN COBOL object code.
SCOBOLLIST	Indicates that you want a SCREEN COBOL compilation listing of the application and identifies a file to which ENABLE is to write this listing.
SCOBOLOBJECT	Identifies the name of the object files for the compiled SCREEN COBOL program.
SCOBOLSKELETON	Identifies the name of the SCREEN COBOL skeleton file.
SCOBOLSOURCE	Indicates that you want a copy of the SCREEN COBOL source program and identifies the name of a file to which ENABLE is to write the source code.
TERMINAL	Identifies the type of terminal on which the application will run.
TITLE	Identifies the text that is displayed on the first screen line by the application.
TREE	Associates boxes with the application.

Table 2-4. Functional Summary of Box Attributes
(Continued next page)

Attribute	Function
BOXTITLE 1 BOXTITLE 2 BOXTITLE 3	Identifies optional text that can appear on the screen with records from a file.
CHECKDATA	Indicates whether the application does, or does not contain special code that verifies the contents of numeric fields in a file.
DATAFILE	Identifies the physical name of a data base file.
DELETE	Indicates whether the application can, or cannot, delete records from a file.
DICTIONARY	Identifies the location of the data dictionary that contains the record description of a file.
EXCLUDE	Identifies fields from the record description that are not to appear on the screen.
FILL	Indicates whether the application does, or does not, perform an automatic read operation on a file under certain conditions.
FLAG	Sets values for user flags defined in the SCREEN COBOL skeleton.
HEADINGS	Identifies the set of labels to be used to identify fields on the screen.
INCLUDE	Identifies fields from the record description that are to appear on the screen; also indicates the order in which the fields appear.

Table 2-4. Functional Summary of Box Attributes (Continued)

Attribute	Function
INSERT	Indicates whether the application can, or cannot, insert records in a file.
NONSTOP	Indicates whether the General Server does, or does not, run as a NonStop process pair.
READ	Indicates whether the application can, or cannot, read records from a file.
RECORD	Identifies the name of the record description that describes a file.
SCREENFORMAT	Identifies the choice of screen layout for a box.
SERVERCLASS	Provides a name for the server class to which the General Server belongs.
SIZE	Identifies the number of records from a file that the application can display at one time.
TMF	Indicates whether a file is, or is not, audited by the Transaction Monitoring Facility (TMF).
UPDATE	Indicates whether the application can, or cannot, update records in a file.
VALUES	Indicates whether the application does, or does not, display initial values from the record description of a file.

ENABLE Commands--Usage Overview

The following paragraphs discuss using the ENABLE commands. Generally, you use these commands to:

- Describe an object
- Add the object to the object table
- Generate an application

Suppose, for example, that you want to generate an application that can access a data base file named "parts." To do this, you:

1. Describe a box that is to represent this file
2. Name the box (for example, "parts-box") and add it to the object table
3. Describe the application that is to use the box
4. Name the application (for example, "parts-in") and add it to the object table
5. Generate the application

Describing an Object

To describe an object, you supply values for the attributes that define the characteristics of that object. ENABLE stores these attribute values in an internal table called the attribute table. This table contains:

- The starting value of each attribute (a value that the attribute has when you start ENABLE)
- The current value of each attribute (a value that you supply with a SET command)
- The override value of the attribute (a temporary value that you supply with an ADD command)
- The default value, if any, of the attribute (a value that ENABLE supplies for certain attributes if the starting value is null and you have not supplied a current or override value)

Figure 2-1 illustrates the attribute table when you start ENABLE and shows the starting values of some attributes; for information about the starting values of all the application and box attributes, refer to the discussion of the SET command in Section 3.

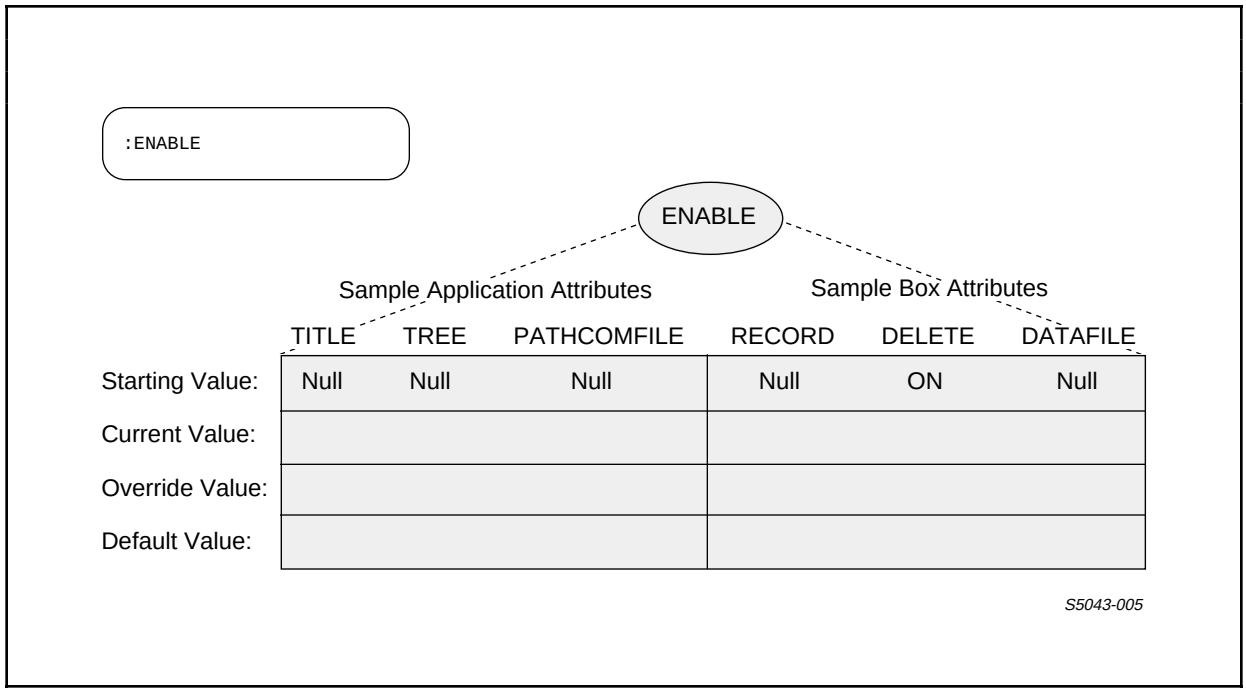


Figure 2-1. Example of Attribute Table When You Start ENABLE

Describing a Box

The value of the box attributes describe a box--the object that represents a data base file used by an application. Suppose, for example, that the record description that describes a file is named "parts." To identify this record description, you must supply a value for the RECORD attribute. You could supply this value by using the following SET command:

```
SET RECORD parts
```

If you enter this SET command, ENABLE stores "parts" as the current value of the RECORD attribute in the attribute table. Figure 2-2 illustrates this process.

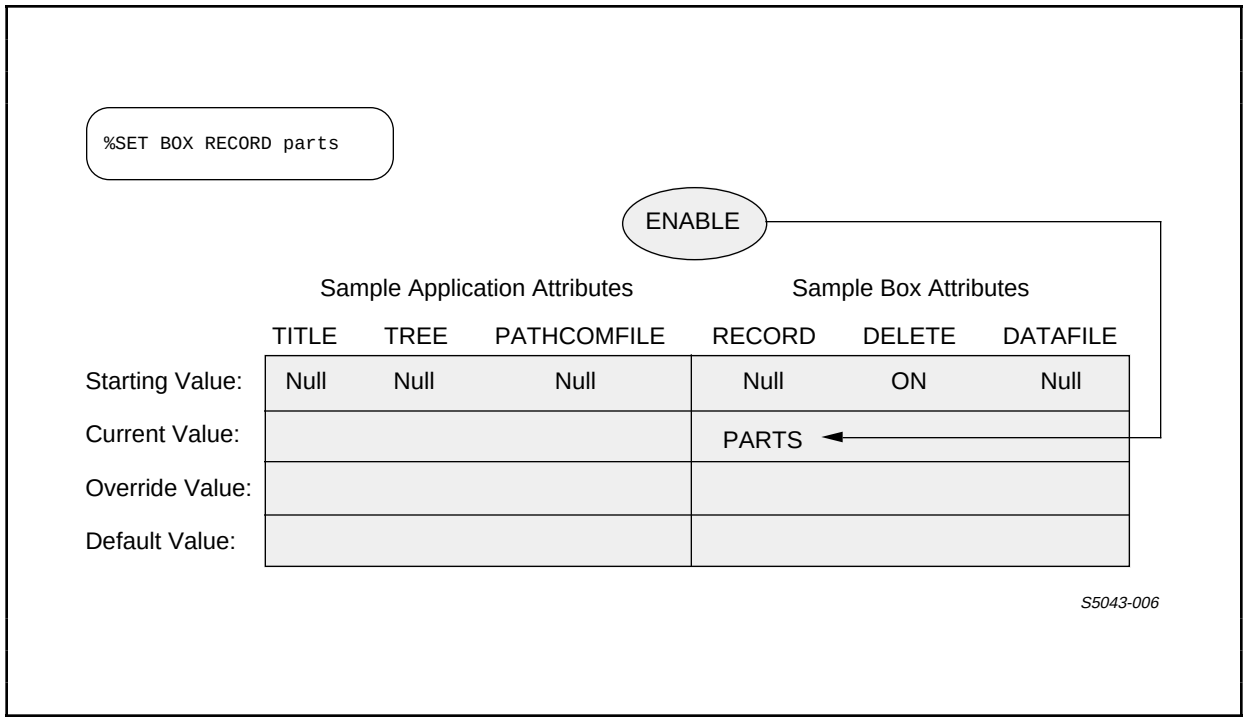


Figure 2-2. Current Value for a Box Attribute

Now suppose that you want to restrict the application to only insert, read, and update operations on a file; that is, you do not want to allow delete operations. You do this by supplying OFF as a value for the DELETE attribute. You could supply this value when you enter the ADD BOX command to name and add the box that represents the file. The following ADD BOX command, for example, supplies OFF as a value for the DELETE attribute:

```
ADD BOX parts-box, DELETE OFF
```

When you supply an attribute value with an ADD BOX command, you supply an override value for the box attribute. An override value is a value that applies only to the object being added. The value supplied for the DELETE attribute, for example, by the preceding ADD BOX command applies only to "parts-box." When you supply an override value with an ADD BOX command, ENABLE enters this value in the attribute table. Figure 2-3 illustrates this process.

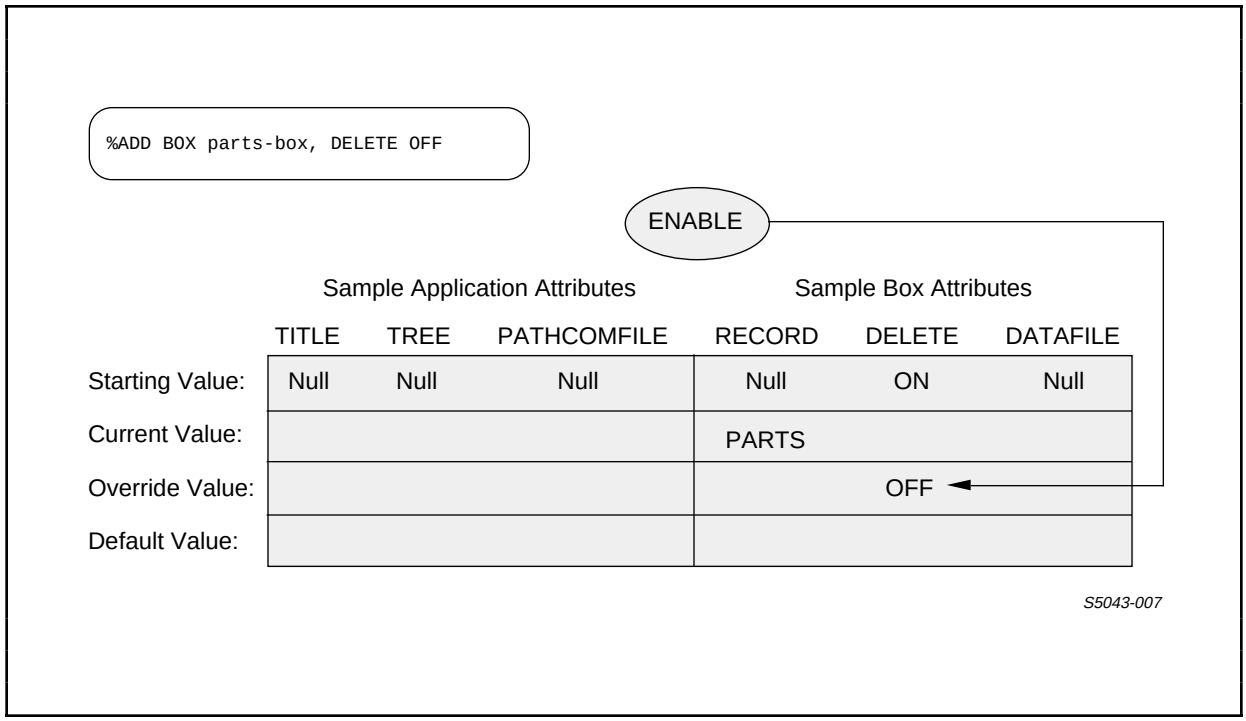


Figure 2-3. Override Value for a Box Attribute

When you enter an ADD BOX command, ENABLE assumes that you have completed your description of a box. ENABLE uses the contents of the attribute table to determine this description. ENABLE evaluates the values in the attribute table as follows:

- An override value, if present, takes precedence over a current or starting value.
- A current value, if present, takes precedence over a starting value.
- If neither an override nor a current value is present, ENABLE uses the starting value.

If you do not supply either a current or override value for some attributes whose starting values are null, ENABLE can supply a default value. ENABLE can, for example, supply a default value for the DATAFILE attribute. ENABLE obtains this value from the record description identified by the value of the RECORD attribute.

Suppose, for example, that you have entered the ENABLE commands described earlier in this section:

```
SET RECORD parts
ADD BOX parts-box, DELETE OFF
```

Figure 2-3 showed the contents of the attribute table after these commands are entered. Since the preceding commands do not supply a value for the DATAFILE attribute, ENABLE will obtain the "parts" record description and enter the file name ("parts") identified in that record description as the default value of the DATAFILE attribute. Figure 2-4 illustrates this process.

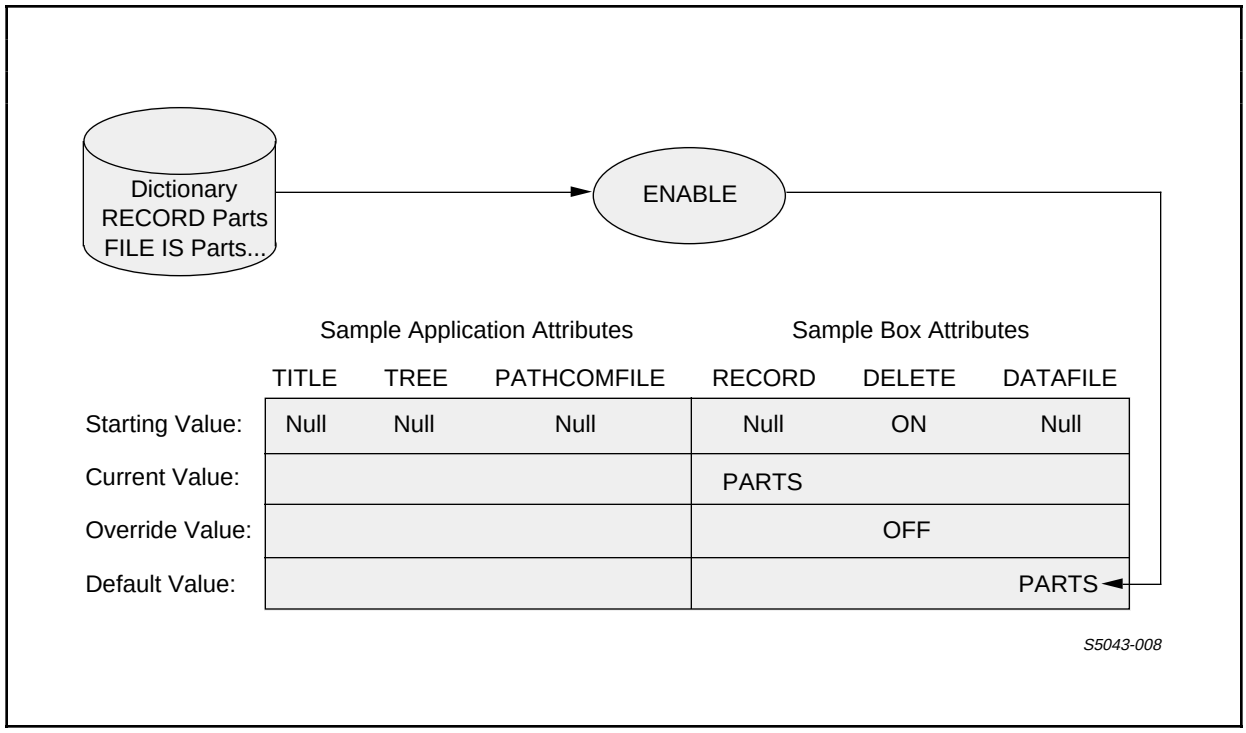


Figure 2-4. Default Value for a Box Attribute

Note that ENABLE supplies a default value at the same time that it enters an override value in the attribute table. After ENABLE has supplied any default values, it adds the box to the object table. This process is discussed later in this section under "ADDING AN OBJECT."

When ENABLE adds a box to the object table, it removes any override or default values for the box attributes from the attribute table. ENABLE does not remove a current value for a box attribute unless you set a new current value with another SET command, enter a RESET command to reset the attribute to its starting value, or exit from ENABLE.

Figure 2-5 illustrates the partial contents of the attribute table after ENABLE adds "parts-box."

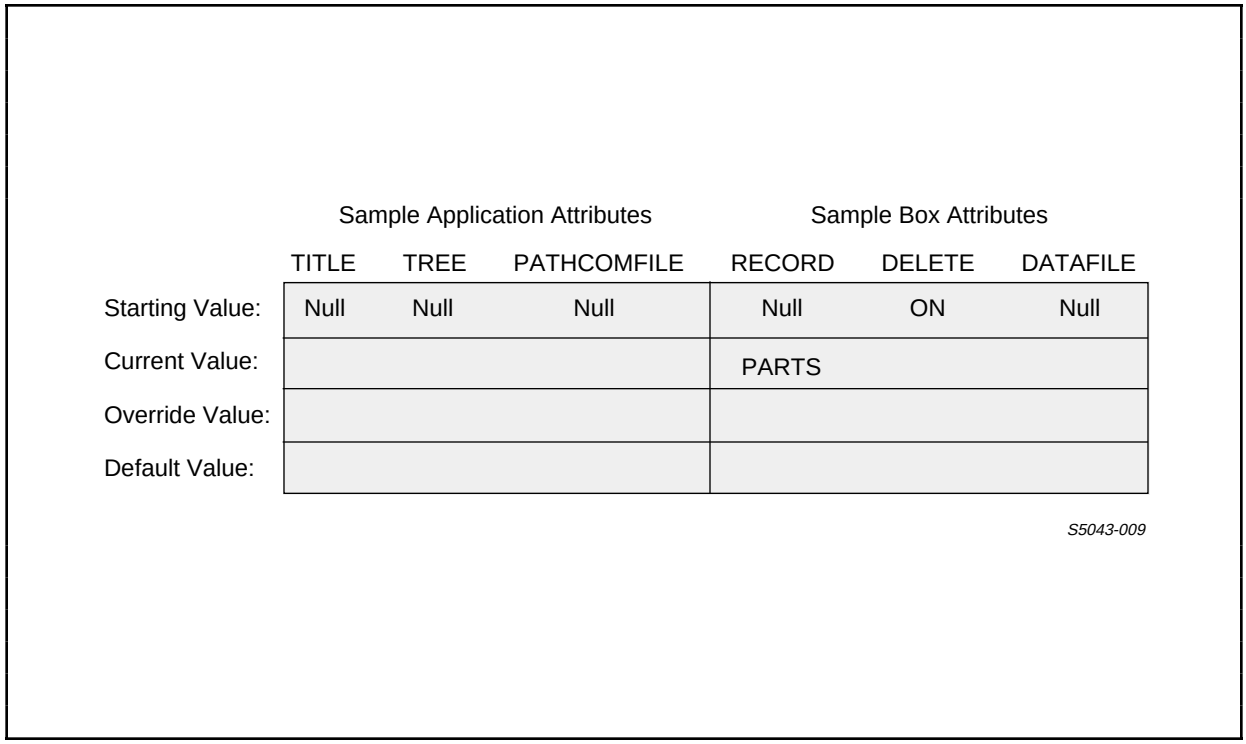


Figure 2-5. Attribute Table After ADD BOX Command

Describing an Application

The values of the application attributes describe global characteristics of an application. Suppose, for example, that you want to identify "enabpath" as the file to which ENABLE is to write the PATHCOM commands for an application. You can identify this file by supplying a value for the PATHCOMFILE attribute. You could supply a value for this attribute by entering the following SET APPL command:

```
SET APPL PATHCOMFILE enabpath
```

When you enter a SET APPL command, you supply a current value for an application attribute. ENABLE enters this current value in the attribute table. Figure 2-6 illustrates this process when ENABLE enters "enabpath" as the current value for the PATHCOMFILE attribute.

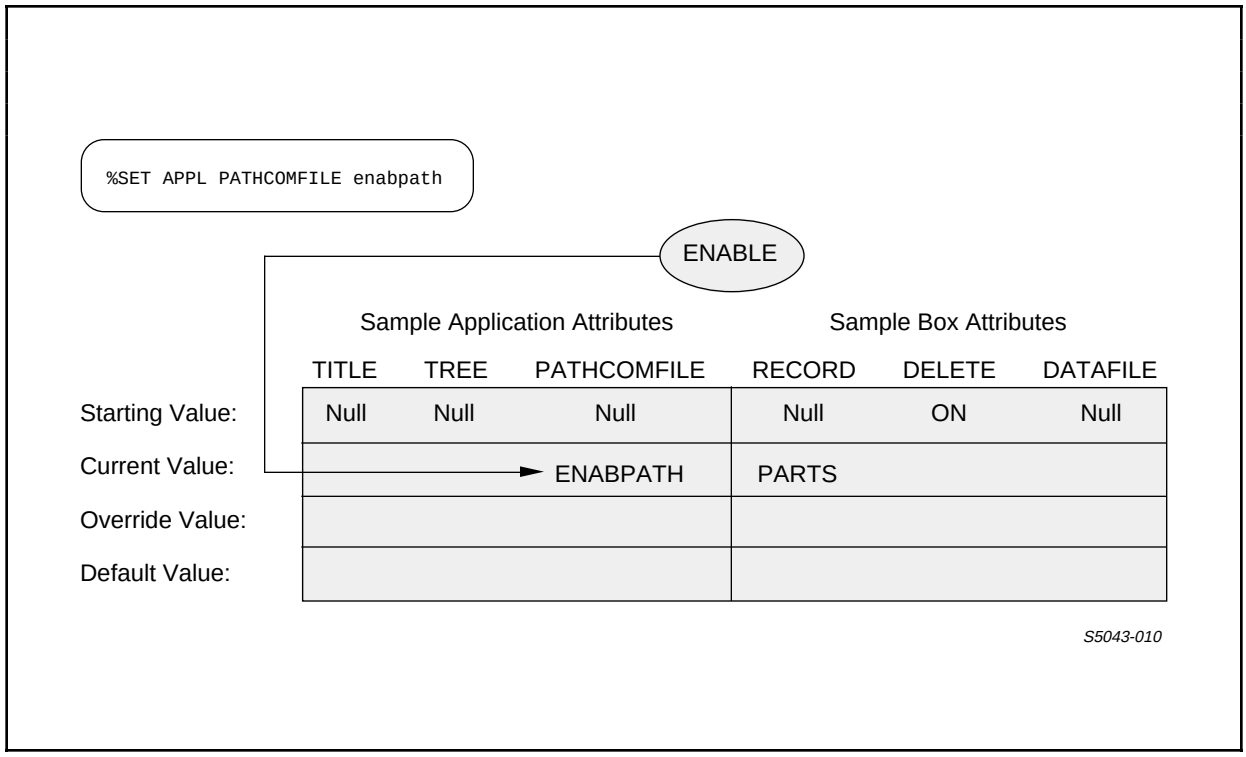


Figure 2-6. Current Value for an Application Attribute

Now suppose that you want to identify "parts-box" as the box to be used by the application. You identify the boxes used by an application by supplying a value for the TREE attribute. You could supply a value for this attribute when you enter the ADD APPL command that names and adds the application. The following ADD APPL command, for example, adds an application called "parts-in" and supplies a value for the TREE attribute:

```
ADD APPL parts-in, TREE (01 parts-box)
```

When you use an ADD APPL command to supply a value for an application attribute, you supply an override value--a value that applies only to the application being added. The override value supplied by the preceding ADD APPL command, for example, only applies to the "parts-in" application. When you supply an override value, ENABLE enters this value in the attribute table. Figure 2-7 illustrates this process.

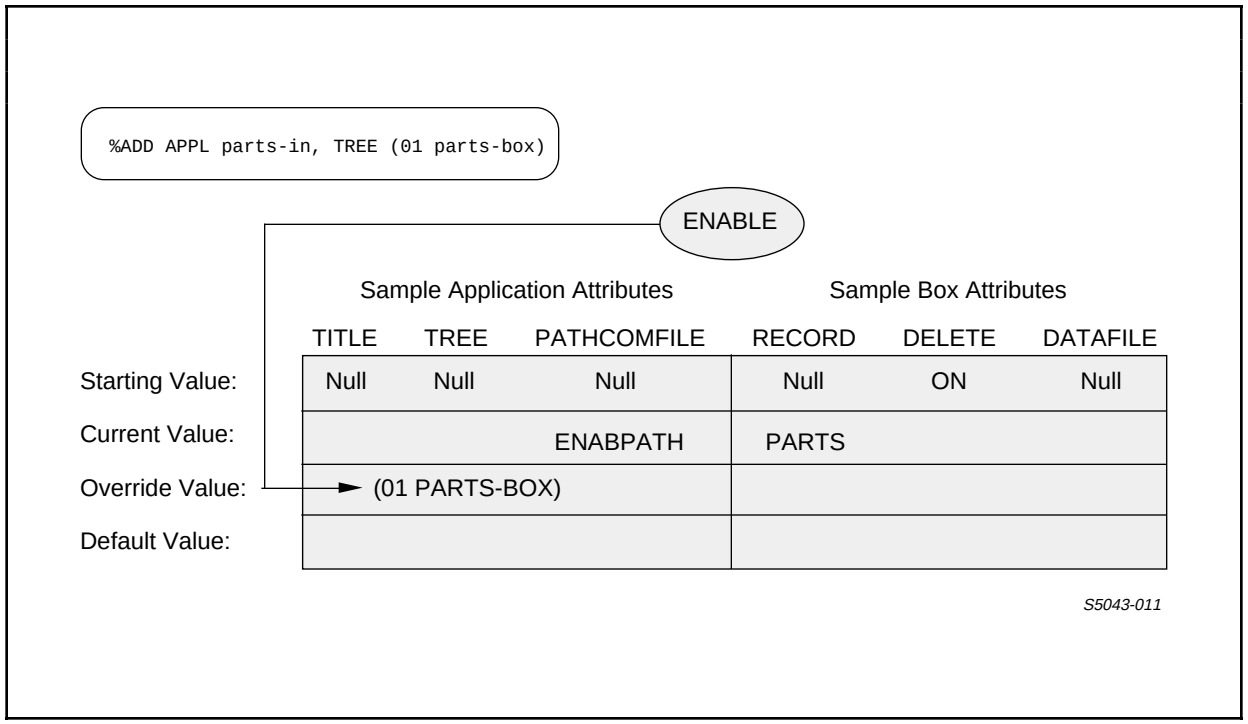


Figure 2-7. Override Value for an Application Attribute

When you enter an ADD APPL command, ENABLE assumes that you have completed your description of the application. ENABLE then examines the contents of the attribute table to determine that description. ENABLE evaluates the values in the attribute table as follows:

- An override value, if present, takes precedence over a current or starting value.
- A current value, if present, takes precedence over a starting value.
- If neither an override nor a current value is present, ENABLE uses the starting value.

If you do not supply either a current or override value, ENABLE supplies a default value for certain application attributes that have null starting values. Suppose, for example, that you have entered the ENABLE commands discussed earlier in this section:

```
SET RECORD parts
ADD BOX parts-box, DELETE OFF
SET APPL PATHCOMFILE enabpath
ADD APPL parts-in, TREE (01 parts-box)
```

If you enter these commands, ENABLE supplies the application name ("parts-in") as a default value for the TITLE attribute. Figure 2-8 illustrates this process.

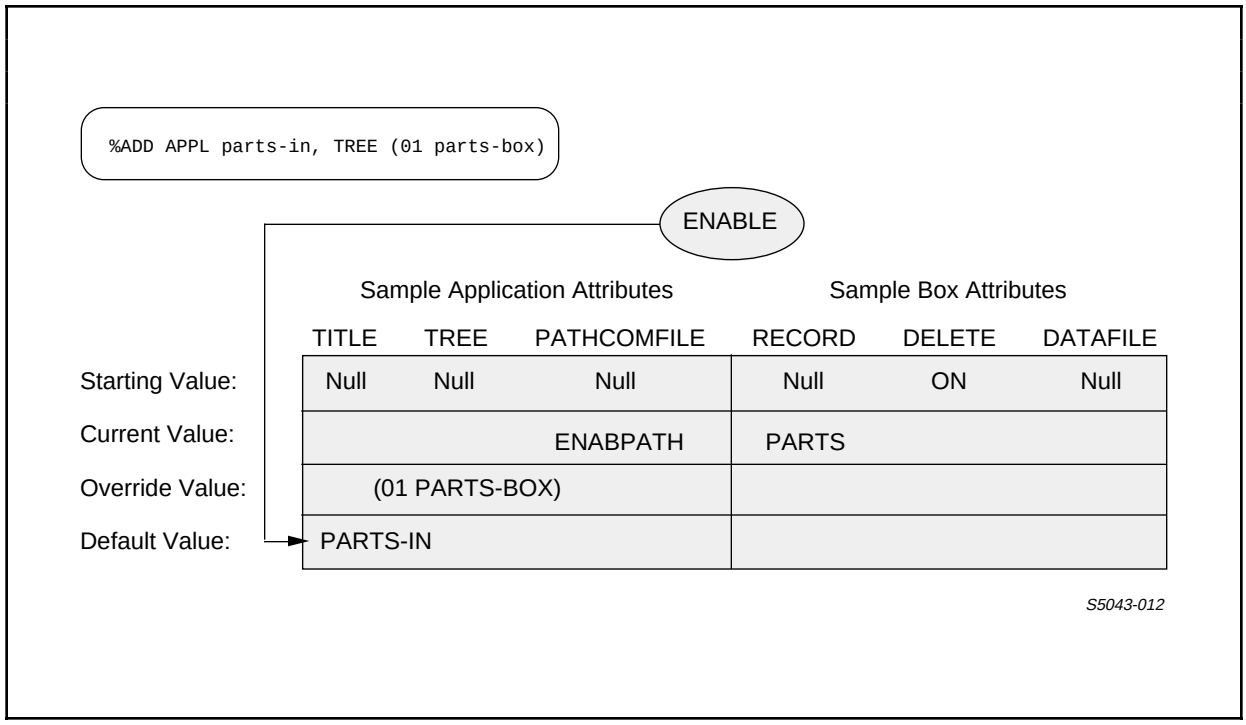


Figure 2-8. Default Value for an Application Attribute

ENABLE supplies the default value of an attribute at the same time that it enters an override value for an attribute.

When ENABLE adds an application to the object table, it removes any override or default values for application attributes. ENABLE does not remove any current values of application attributes. These values remain in the attribute table until you supply a new current value with another SET APPL command, reset the attribute to its starting value with a RESET command, or exit from ENABLE.

Adding an Object

To add an object to the object table explicitly, you must use the ADD command. Under certain conditions, ENABLE implicitly adds an object to the object table for you. (Section 4 describes these conditions.) In some cases, implicitly adding an object can lead to unexpected results; to avoid this problem, explicitly add each object with an ADD command.

When you enter an ADD BOX command, ENABLE examines the contents of the attribute table, determines the description of the box, and adds the box to the object table. Figure 2-9 illustrates this process.

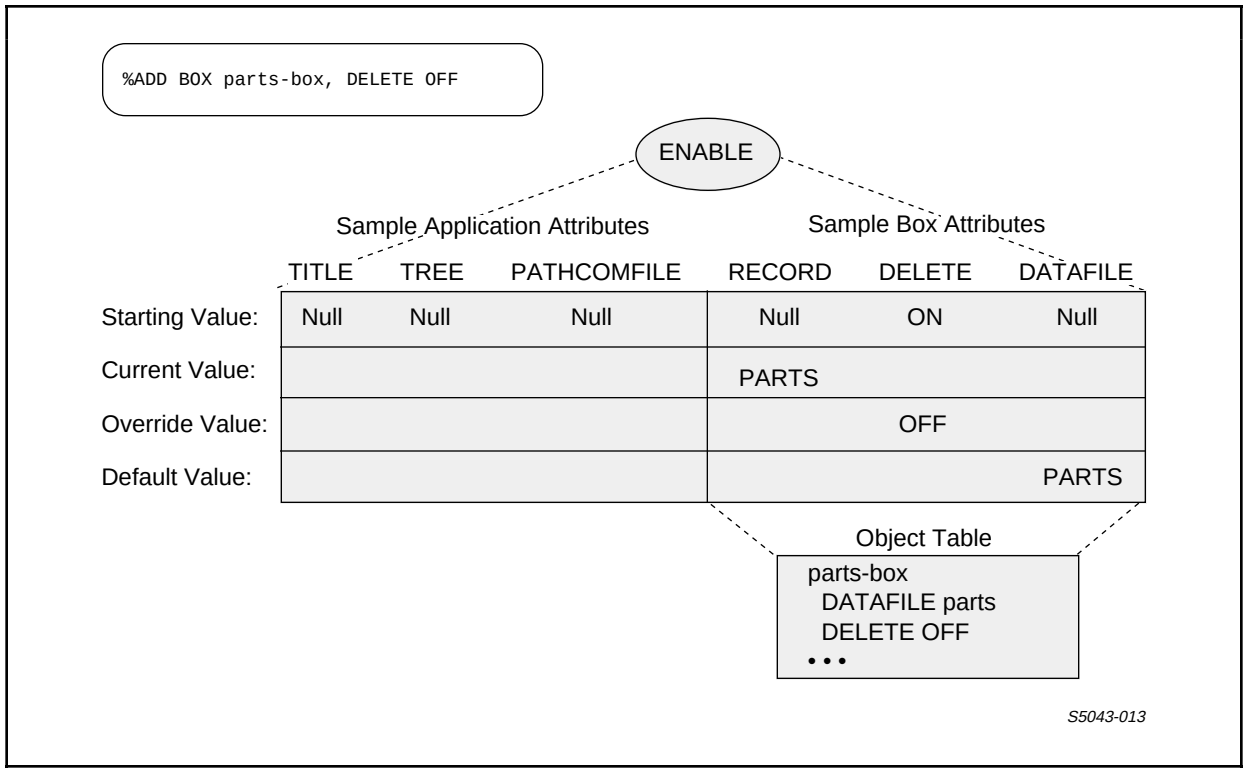


Figure 2-9. Effect of ADD BOX Command

When you enter an ADD APPL command, ENABLE examines the contents of the attribute table, determines the description of the application, and adds the application to the object table. Figure 2-10 illustrates this process.

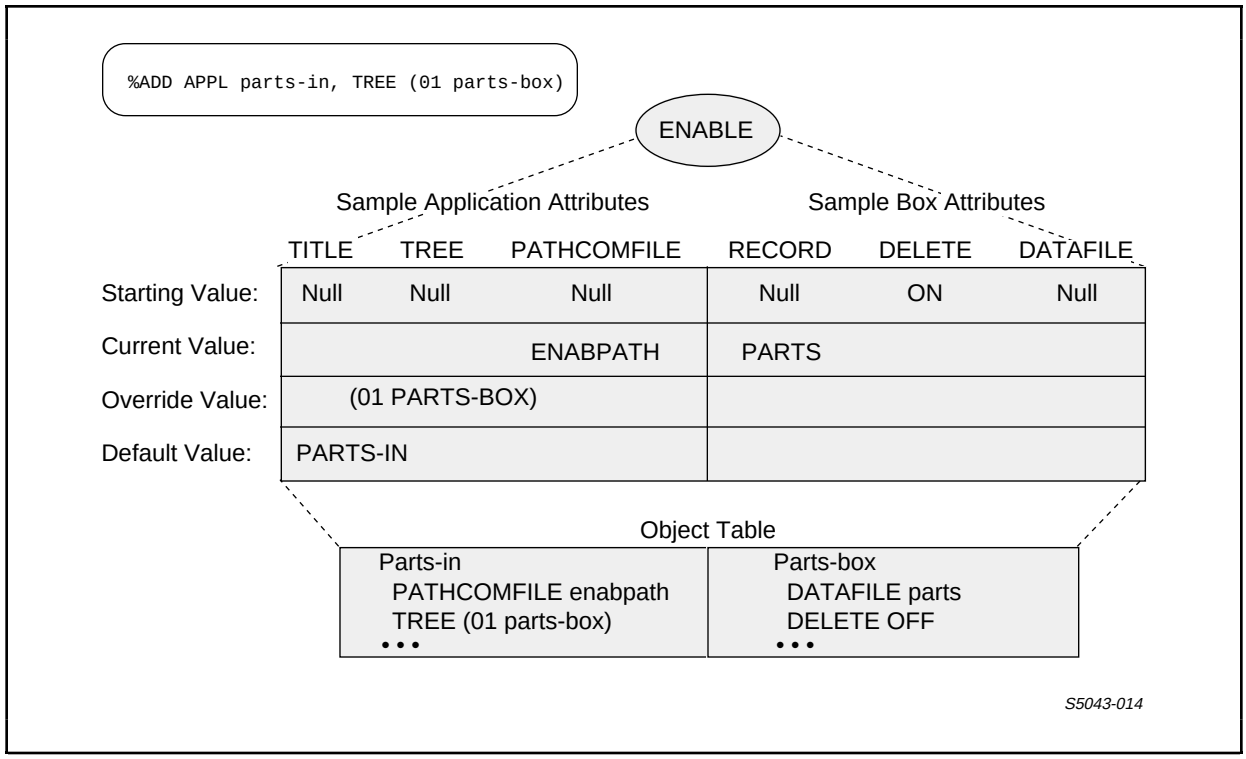


Figure 2-10. Effect of an ADD APPL Command

Once ENABLE has added an object to the object table, you cannot change the description of the object. You can, however, use the DELETE command to delete the object; use the SET, RESET, or ADD command to supply new values for the object; and replace the object with its new description by using the ADD command.

Generating an Application

To indicate that you want to generate an application, you use the GENERATE command. When you enter this command, the ENABLE compiler passes the contents of the object table to the program generator. The program generator uses the description of the application (specifically, the value of the TREE attribute) to determine which boxes will be used by the application and then generates the application.

ENABLE COMMAND CONVENTIONS

The following paragraphs describe some of the conventions that apply to the ENABLE commands.

Multiline Commands

Two or more commands can be grouped and separated by semicolons (;):

```
%command-1; command-2
```

is equivalent to:

```
%command-1
%command-2
```

Command Line Continuation

Generally, the maximum length of an ENABLE command line is 132 characters. You can, however, continue a command line by including the ENABLE continuation character, the ampersand (&):

```
SET BOXTITLE 1 "This boxtitle text is continued on to the ne&
xt command line"
```

The maximum length of a continued command line is 528 characters.

Three ENABLE commands (SET INCLUDE, SET EXCLUDE, and SET TREE) include parentheses as part of their syntax. You can continue the portion of these commands within the parentheses onto any number of subsequent lines without specifying a continuation character.

Comments

Comments are useful only as documentation within a command file. You can include comments with all commands. Comments must be delimited by the double hyphen (--) symbol. ENABLE ignores the double hyphen symbol and all subsequent characters up to the next double hyphen or the end of the effective command line, for example:

```
SET DELETE OFF -- Prohibits delete operations
ADD BOX parts-box, RECORD parts, -- 2 records in box -- SIZE 2
```

Reserved Words

Reserved words are words with special meaning to ENABLE. You cannot use reserved words as field names, box names, or application names. Appendix C provides a list of the reserved words.

Field Name Qualification

To avoid ambiguity, several of the ENABLE commands require you to use unique field names. If a field name is not unique, you must qualify it. The syntax used to qualify a field name is as follows:

```
{ <group-name> [ .<group-name> ... ] .<field-name> }
{                                                  }
{ KEY                                              }
```

<group-name>

is the name of a group in the record description of which <field-name> is a part. A group is an item in a record description whose level number (01, 02, ...) is higher than the level number of the following item; a level number of 01 is considered to be the highest level number.

If necessary, you can qualify <field-name> by including more than one <group-name>.

<field-name>

is the name of the field requiring qualification.

KEY

is the courtesy key (Record Number field) of a relative, entry-sequenced, or unstructured file. For a key-sequenced file, KEY identifies the primary key.

Consider the record description shown in Figure 2-11.

```
RECORD events.
FILE IS event  KEY-SEQUENCED.
02 event-key   ...
02 event-desc  ...
02 predicted-dates.
   04 starting.
      06 day   ...
      06 month ...
      06 year  ...
    04 ending.
      06 day   ...
      06 month ...
      06 year  ...
02 actual-dates.
   04 starting.
      06 day   ...
      06 month ...
      06 year  ...
   04 ending.
      06 day   ...
      06 month ...
      06 year  ...
KEY 0 IS event-key.
END
```

Figure 2-11. Sample Record Description

This record description contains several groups; for example: "predicted-dates" and "actual-dates." Nested within both of these groups are other groups: "starting" and "ending." If you want to refer to the field named year indicated in the following portion of this record description:

```
02 predicted-dates.
   04 starting.
      06 day   ...
      06 month ...
      06 year  ...
   04 ending.
      06 day   ...
      06 month ...
      06 year  ...  <-------- the ending year of the
                              predicted dates
```

you must enter:

```
predicted-dates.ending.year
```

To refer to the primary key ("event-key") of the "events" record description, you could enter either:

```
KEY or event-key
```

String Literals

Several of the ENABLE commands allow you to specify a string literal. A string literal consists of one or more characters enclosed within quotation marks. You can specify any character, including a blank, within a string literal. Consider the following examples of string literals:

```
"This is a string literal"
" "
"Part Name            Part Number"
```

SECTION 3

ENABLE COMMANDS

ENABLE commands permit you to control and describe objects within the ENABLE subsystem. They allow you to specify attributes that define two types of objects: applications and boxes. Refer to Section 2 for more information about these objects.

This section describes the syntax of the ENABLE commands and the function that each command performs. Generally, you use the ENABLE commands as follows:

1. Use the SET command to establish values for the attributes that describe the characteristics of an object.

2. Use the ADD command to name the object and add its description to the object table.

3. Use the GENERATE command to generate the application after you have added the associated application.

Refer to Figure 3-1 for specific guidelines to follow when you use the ENABLE commands to generate a single-file application.

1. Use the SET BOX RECORD command to set a value for the box attribute that identifies the record description of the data base file.

2. Optionally, use the SET BOX command to describe other attributes of the box.

3. Use the ADD BOX command to name the box that represents the data base file and to add the description of the box to the object table.

4. Use the SET APPL TREE command to set a value for the application attribute that associates the box just added with the application.

5. Use the SET APPL PATHCOMFILE command to set a value for the application attribute that identifies the name of the PATHCOM command file.

6. Optionally, use the SET APPL command to set a value for any other attributes of the application.

7. Use the ADD APPL command to name the application and add it to the object table.

8. Use the GENERATE command to generate the added application.

Figure 3-1. Recommended Command Order for Generating a Single-File Application

Figure 3-2 shows the ENABLE commands used to generate a sample single-file application. This figure also shows the terminal screen displayed by the application.

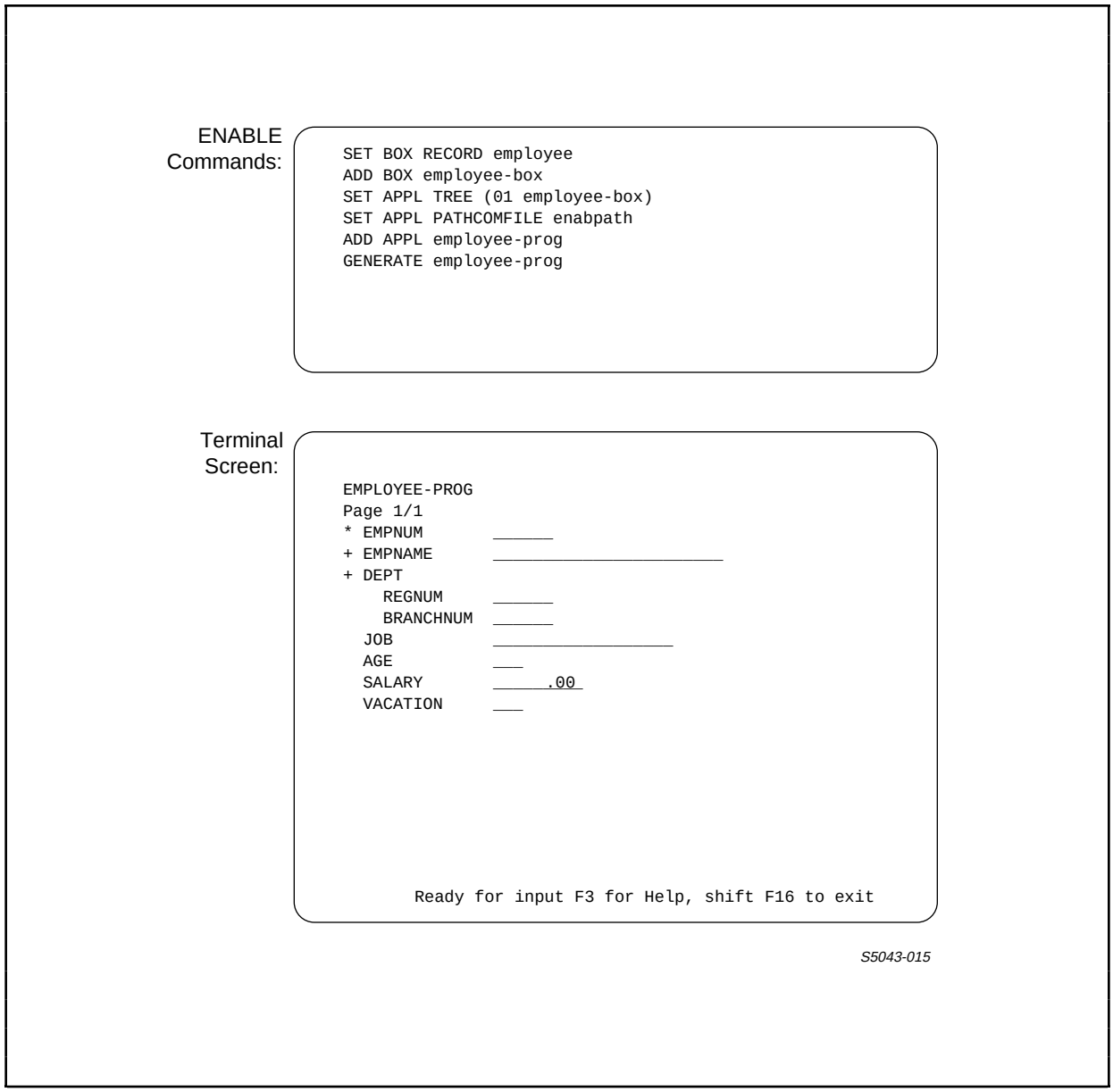


Figure 3-2. Sample Commands for a Single-File Application

Figure 3-3 provides specific guidelines to follow when you use the ENABLE commands to generate a multifile application.

1. Use the SET BOX RECORD command to set a value for the box attribute that identifies the name of the record description associated with a particular data base file.

2. Optionally, use the SET BOX command to set a value for the other box attributes.

3. Use the ADD BOX command to name the box that represents a particular data base file and to add the description of the box to the object table.

4. Use the RESET BOX command to reset the values of the box attributes before you add the next box.

5. Repeat steps 1 through 4 until you have added all the boxes necessary for the application.

6. Use the SET APPL TREE command to set a value for the application attribute that associates the added boxes with the application.

7. Use the SET APPL PATHCOMFILE command to set a value for the application attribute that identifies the name of the PATHCOM command file.

8. Optionally, use the SET APPL command to set the value of any other attributes that describe the application.

9. Use the ADD APPL command to name the application and add it to the object table.

10. Use the GENERATE command to generate the added application.

Figure 3-3. Recommended Command Order for Generating a Multifile Application

Since some of the commands necessary to generate a multifile application are rather lengthy, you would normally enter these commands in an edit-type obey file for submission to ENABLE.

Figure 3-4 shows the ENABLE commands used to generate a sample multifile application. This figure also shows the terminal screen displayed by the application.

ENABLE Commands:

```
SET BOX RECORD parts
SET BOX BOXTITLE 1 "Parts Box:"
ADD BOX parts
RESET BOX *
SET BOX RECORD inventory
ADD BOX inventory
SET APPL TREE (01 parts
  02 inventory LINK partnum TO OPTIONAL partno)
SET PATHCOMFILE multipath
ADD APPL stock-control
GENERATE APPL stock-control
```

Terminal Screen:

```
STOCK-CONTROL
Page 1/1
Parts Box:
* PARTNUM   _____

  * PRIMKEY
     LOCATION-NUM   _______
   QUANTITY-ON-HAND _______
   REORDER-LEVEL    _______

+ PARTNAME  ________________________
  PRICE     _______.00

     Ready for input   F3 for Help, shift F16 to exit
```

S5043-016

Figure 3-4. Sample Commands for a Multifile Application

ENABLE commands are described on the pages that follow. The syntax and function are presented for each command in alphabetical order.

ADD Command

The ADD command names an object and enters its description in the object table. Optionally, the ADD command also supplies override values for the attributes that describe an object. The syntax of the ADD command is:

```
ADD [ APPL ] <object> [ , LIKE <prior-object> ]
    [ BOX  ]

  [ , <attribute> <value> ] ...
```

APPL

is a keyword that identifies application as the type of object being added. If APPL is the default object type, you can omit this keyword.

BOX

is a keyword that identifies box as the type of object being added. If BOX is the default object type, you can omit this keyword.

<object>

is the name of an object to be added. This name must conform to the following naming conventions:

1. The name can have from 1 to 30 letters, digits, and hyphen characters. It must begin with a letter and must not end with a hyphen.

2. The name must not be an ENABLE, SCREEN COBOL, or PATHWAY reserved word (listed in Appendix C).

3. The name must not begin with the 6-character ENABLE prefix "T9155-."

4. The name must be unique. Duplicate object names are not allowed.

--->**>

5. For an application name, the name must not be a PATHCOM reserved word when truncated to 15 characters.

6. For a box name, the name must not be the same as any field name in the record description associated with that box.

LIKE <prior-object>

sets the attributes of the object to be added to those of <prior-object>. When you use this form of the ADD command, <prior-object> must appear in the object table and be of the same type as <object>. ENABLE ignores values in the attribute table when you use the LIKE clause.

<attribute>

is either an application attribute, or a box attribute. When BOX is the object type in effect you can specify only box attributes. When APPL is in effect you can specify both application and box attributes. Application and box attributes are classified as follows:

Application Attributes:

PATHCOMFILE	SCOBOLOBJECT	TITLE
PATHCOMSKELETON	SCOBOLSKELETON	TREE
SCOBOLCOMPILER	SCOBOLSOURCE	
SCOBOLLIST	TERMINAL	

(You cannot specify application attributes when BOX is the object type in effect.)

Box Attributes:

BOXTITLE	EXCLUDE	INSERT	SERVERCLASS
CHECKDATA	FILL	NONSTOP	SIZE
DATAFILE	FLAG	READ	TMF
DELETE	HEADINGS	RECORD	UPDATE
DICTIONARY	INCLUDE	SCREENFORMAT	VALUES

Application and box attributes are described in Section 4.

-->

<value> is any legal value for the specified attribute. When you supply a value for an attribute with an ADD command, you override any value set for the same attribute with a SET command or a LIKE clause. The value only applies to the object being added; this command does not change the current value of the attribute in the attribute table.

Considerations

Unless specified within the ADD command itself, ENABLE uses values from the attribute table to specify characteristics of the object being added.

Once entered in the object table, the description cannot be modified directly. You can, however, remove an object using the DELETE command, specify new values with the SET command, and enter the object once again with the ADD command. Refer to the DELETE command description, later in this section, for more information about removing applications and boxes from the object table.

ENABLE issues a message when the ADD command is successful:

```
Application added:  <application-name>
```

or:

```
Box added:  <box-name>
```

ENABLE issues an error message if it cannot add an object.

Adding a Single-File Application. To preserve compatibility with previous versions of ENABLE, a single-file application can be added based on the current value of the RECORD box attribute.

You do not have to add a box to represent the file or supply a value for the TREE attribute for a single-file application. If you do not supply a value for the TREE attribute, ENABLE:

- Appends the characters -BOX to the application name to form a box name
- Uses (01 <application-name>-BOX) as the default value of the TREE attribute

Adding a Multifile Application. Before you can add a multi-file application, you must:

- Use the ADD BOX command to add every box that represents a data base file to be accessed by the application
- Use the SET command to set a value for the TREE attribute. ENABLE uses the value of this attribute to organize the boxes within the application

Adding a Box. To name and add a box, you must first supply a value for the RECORD attribute. If you do not supply a value for the RECORD attribute, ENABLE cannot add the box.

When you generate a multifile application, you must add a box for every data base file to which the application is to have access.

ENABLE issues an error message if it cannot add a box. It issues the following informative message when it adds a box:

Box added: <box-name>

Examples

Consider the ADD commands in the following series of ENABLE commands:

```
SET RECORD parts
ADD BOX parts  <------------------  adds a box named
   Box added:  PARTS                "parts"

SET APPL PATHCOMFILE newpart
SET APPL TREE (01 parts)
ADD APPL parts-info  <------------  adds an application
   Application added:  PARTS-INFO   named "parts-info" to
                                    the object table
```

The following ADD BOX command adds a box and supplies a value for the RECORD attribute:

```
ADD BOX parts, RECORD parts  <----  adds a box named "parts"
                                    and supplies a value for
                                    its RECORD attribute
```

The ADD APPL command in the following sequence of ENABLE commands adds an application and supplies a value for the TREE application attribute:

```
SET RECORD parts
ADD BOX parts
SET RECORD inventory
ADD BOX inventory
SET PATHCOMFILE multpath !
ADD APPL inventory, TREE (01 parts  <------  adds an applic-
                          02 inventory      ation named
                          LINK partnum      "inventory" and
                          TO OPTIONAL       supplies a value
                          partno)           for its TREE
                                            attribute
```

ASSUME Command

The ASSUME command establishes a default object type for subsequent ENABLE commands. The default object type applies to any command that omits the keywords: APPL or BOX. The syntax of the ASSUME command is:

```
ASSUME { APPL }
       { BOX  }

APPL

  is a keyword that identifies APPL as the default object
  type.  ENABLE treats subsequent commands that omit an
  object type as if they include the keyword APPL.

BOX

  is a keyword that identifies BOX as the default object
  type.  ENABLE will treat subsequent commands that omit an
  object type as if they include the keyword BOX.
```

Consideration

Since APPL is the default object type when you start ENABLE, you do not need to use the ASSUME APPL command unless you want to reset the default object type after entering an ASSUME BOX command.

Example

Consider the following series of commands:

```
ASSUME BOX  <-------------------  establishes BOX as the
                                  default object type for
                                  subsequent commands
SET RECORD employee
SET SCREENFORMAT COMPRESSED
ADD employ  <-------------------  adds a box named "employ"

ASSUME APPL  <------------------  establishes APPL as the
                                  default object type for
                                  subsequent commands
```

DELETE Command

The DELETE command removes a previously added object from the object table. The syntax of the DELETE command is:

```
DELETE [ APPL ] { <object-name> }
       [ BOX  ] { *             }
```

APPL

is a keyword that identifies application as the type of object being deleted. If APPL is the default object type, you can omit this keyword when you delete an application.

BOX

is a keyword that identifies box as the type of object being deleted. If BOX is the default object type, you can omit this keyword when you delete a box.

<object-name>

is the name of an object to be deleted from the object table.

*

indicates that all objects of the type in effect are to be deleted from the object table.

Considerations

The DELETE APPL command deletes applications; it does not delete any of the boxes associated with an application. To delete a box, you must use the DELETE BOX command.

If you attempt to delete an object not listed in the object table, ENABLE issues an error message.

A box that is named in the TREE value of an application cannot be deleted until all applications that name it have been deleted. If you attempt to delete a box that is

associated with an application entry, ENABLE issues an error message.

Deleting All Objects. If ENABLE issues a warning message stating:

> Very low on extended memory; please DELETE unwanted objects

you might want to delete some objects and then generate the remaining applications. After you have generated these applications, you can delete all the objects in the object table by entering:

```
DELETE APPL *
DELETE BOX *
```

When you issue these commands, ENABLE reclaims all space used in the object table for the deleted applications and boxes.

Example

The following series of commands deletes an application and its associated boxes:

```
DELETE APPL parts-info  <---  deletes an application named
                              "parts-info"
DELETE BOX parts  <---------  deletes a box named "parts"
DELETE BOX supplier  <------  deletes a box named
                              "supplier"
```

GENERATE Command

The GENERATE command causes ENABLE to generate one or more applications. The syntax of the GENERATE command is:

```
GENERATE [ APPL ] [ <application> ]
                  [ *             ]

  [ , <attribute> <value> ] ...
```

APPL

is a keyword that identifies application as the type of object being generated. If APPL is the default object type, you can omit this keyword when you generate an application.

<application>

is the name of the application to be generated. Application names must conform to the following naming conventions:

1. The name can have from 1 to 30 letters, digits, and hyphen characters. It must begin with a letter and must not end with a hyphen.

2. The name must not be an ENABLE, SCREEN COBOL, or PATHWAY reserved word (listed in Appendix C) and must not be a PATHCOM reserved word when truncated to 15 characters.

3. The name must not begin with the 6-character ENABLE prefix "T9155-."

4. The name must be a unique entry to the object table. Duplicate object names are not allowed.

*

indicates that all named applications (those that currently exist in the object table) are to be generated.

--->**>

<attribute>

is either an application attribute, or a box attribute. When BOX is the object type in effect you can specify only box attributes. When APPL is in effect you can specify both application and box attributes. Application and box attributes are classified as follows:

Application Attributes:

PATHCOMFILE	SCOBOLOBJECT	TITLE
PATHCOMSKELETON	SCOBOLSKELETON	TREE
SCOBOLCOMPILER	SCOBOLSOURCE	
SCOBOLLIST	TERMINAL	

(You cannot specify application attributes when BOX is the object type in effect.)

Box Attributes:

BOXTITLE	EXCLUDE	INSERT	SERVERCLASS
CHECKDATA	FILL	NONSTOP	SIZE
DATAFILE	FLAG	READ	TMF
DELETE	HEADINGS	RECORD	UPDATE
DICTIONARY	INCLUDE	SCREENFORMAT	VALUES

Application and box attributes are described in Section 4.

<value>

is any legal value for the specified attribute.

Considerations

Values for attributes supplied with the GENERATE command override values previously set for those attributes, with the following exceptions:

- If you have previously added the application to the object table, ENABLE ignores any values for application attributes you supply with the GENERATE command.

- If the application uses a box or boxes that you have previously added to the object table, ENABLE ignores values for box attributes supplied with a GENERATE command.

If you use the following form of the GENERATE command:

```
GENERATE APPL *
```

without previously adding any applications, ENABLE ignores the command.

If you enter a GENERATE command and <application-name> is not a current object-table entry, ENABLE:

- Performs an ADD APPL using the values in the attribute table, and any override values you supply with the GENERATE command
- Performs an ADD BOX if ENABLE has supplied a default value for the TREE attribute and the box identified by this default value is not listed in the object table (If the box appears in the object table, ENABLE uses the existing box.)
- Generates the application
- Performs a DELETE APPL to delete the application it just added
- Performs a DELETE BOX if it added a box for the application (ENABLE will not delete a box that you have entered.)

Examples

```
SET RECORD employee
ADD BOX employee-box
SET APPL TREE (01 employee-box)
SET APPL PATHCOMFILE enabfile
GENERATE APPL employee-prog  <-  These commands add an
                                 application named
                                 "employee-prog," cause the
                                 application to be generated,
                                 and delete the application
                                 from the object table.
```

The following GENERATE command supplies values for the RECORD and PATHCOMFILE attributes and generates a single-file application named new-parts:

```
GENERATE APPL new-parts, RECORD parts, PATHCOMFILE partpath
```

INFO Command

The INFO command displays contents of the object table. The syntax of the INFO command is:

```
INFO [ APPL ] { <object> } [ , BRIEF  ]
     [ BOX  ] { *        } [ , DETAIL ]
```

APPL

is a keyword that identifies application as the type of object for which values are to be displayed. If APPL is the default object type, you can omit this keyword when you enter the INFO command.

BOX

is a keyword that identifies box as the type of object whose attribute values are to be displayed. If BOX is the default object type, you can omit the keyword BOX when you enter the INFO command.

<object>

is the name of an object that appears in the object table.

*

indicates that the INFO command applies to all of the listed objects of the named or default object type.

BRIEF

If APPL is the object type, ENABLE displays the application name (or names if you used the * option), along with the boxes associated with that application (as determined by the value of the TREE attribute). The box names are indented following the application name.

If BOX is the object type, ENABLE displays the box name (or names if you used the * option).

-->

DETAIL

If APPL is the object type, ENABLE displays:

- The application name (or names if you used the * option)
- All application attributes and the values that were in effect when the application was added, including any starting, default, and null values
- The names of all boxes listed in the TREE attribute
- Under each box name, all box attributes and values in effect when the box was added, including any starting, default, and null values

If BOX is the object type, ENABLE displays:

- The box name (or names if you used the * option)
- Under the box name, all box attributes and values in effect when the box was added, including any starting, default, and null values

If you omit both the BRIEF and DETAIL options, ENABLE displays the object's name and the non-null attributes that describe it.

Consideration

If the named object does not appear in the object table, ENABLE ignores this command.

Examples

To display a list of the attributes that describe an application, include the application name when you enter the INFO APPL command option; for example, if an application named "emp-program" has been added, the command:

```
INFO APPL emp-prog
```

causes ENABLE to display:

```
-----------------------------------------------------
Application                  EMP-PROG
-----------------------------------------------------
Pathcomfile          $XYZ.PDQ.MYPATH
Pathcomskeleton      $SYSTEM.SYSTEM.ENABPATS
SCOBOLobject         $XYZ.PDQ.POBJ
SCOBOLskeleton       $SYSTEM.SYSTEM.ENABAPPS
Terminal             T16-6520
Title                EMP-PROG
Tree                 (01 EMPLOYEE
                       02 DETAILS
                       Link EMPNUM to optional EMPNO )
```

To obtain a list of the boxes associated with an application, enter:

```
INFO APPL emp-prog, BRIEF
```

ENABLE displays:

```
Application          EMP-PROG
  Box                EMPLOYEE
  Box                DETAILS
```

To obtain information about the attribute values of a named box, enter:

```
INFO BOX parts, DETAIL
```

To obtain a list of all boxes in the object table, enter:

```
INFO BOX *, BRIEF
```

RESET Command

The RESET command resets the current value of an attribute (or set of attributes) to its starting value. The syntax of the RESET command is:

```
RESET [ APPL ] { [ <attribute> ]       }
      [ BOX  ] { [ ABILITY     ]       }
               { [ FORMAT      ]       }
               { [ INTEGRITY   ]       }
               { [ OTHER       ] , ... }
               { *                     }
```

APPL

is a keyword that identifies application as the type of attribute being reset. If APPL is the default object type, you can omit this keyword when you reset the value of an application attribute.

BOX

is a keyword that identifies box as the type of attribute being reset. You can omit this keyword when you reset the value of a box attribute.

<attribute>

is either an application attribute, or a box attribute. When BOX is the object type in effect you can specify only box attributes. When APPL is in effect you can specify both application and box attributes. Application and box attributes are classified as follows:

Application Attributes:

PATHCOMFILE	SCOBOLOBJECT	TITLE
PATHCOMSKELETON	SCOBOLSKELETON	TREE
SCOBOLCOMPILER	SCOBOLSOURCE	
SCOBOLLIST	TERMINAL	

(You cannot specify application attributes when BOX is the object type in effect.)

--->**>

Box Attributes:

BOXTITLE	EXCLUDE	INSERT	SERVERCLASS
CHECKDATA	FILL	NONSTOP	SIZE
DATAFILE	FLAG	READ	TMF
DELETE	HEADINGS	RECORD	UPDATE
DICTIONARY	INCLUDE	SCREENFORMAT	VALUES

Application and box attributes are described in Section 4.

ABILITY

indicates that ENABLE is to reset the values of a set of attributes that determine the ability of an application to delete, insert, read, or update a data base file. These attributes are: DELETE, FILL, INSERT, READ, and UPDATE.

FORMAT

indicates that ENABLE is to reset the values of a set of attributes that determine the screen format. These attributes are: BOXTITLE 1, BOXTITLE 2, BOXTITLE 3, EXCLUDE, HEADINGS, INCLUDE, SCREENFORMAT, SIZE, and VALUES.

INTEGRITY

indicates that ENABLE is to reset the values of a set of attributes that determine the method the application uses to ensure the integrity of a data base file. These attributes are: CHECKDATA, NONSTOP, and TMF.

OTHER

indicates that ENABLE is to reset the values of the following attributes: DATAFILE, DICTIONARY, FLAG, RECORD, and SERVERCLASS.

-->

*
If APPL is object type, this symbol indicates that ENABLE is to reset all box and application attributes to their starting values.
If BOX is the object type, this symbol indicates that ENABLE is to reset all box attributes to their starting values.

Considerations

In order to maintain compatibility with previous versions of ENABLE, the RESET APPL * command affects both box and application attributes. Figure 3-5 illustrates this point.

When you enter the RESET APPL * command, ENABLE issues the message:

```
Box attributes were reset
```

ENABLE issues this message to remind you that it reset the box attributes along with the application attributes.

If you enter the RESET BOX * command, ENABLE resets all box attributes to their starting values.

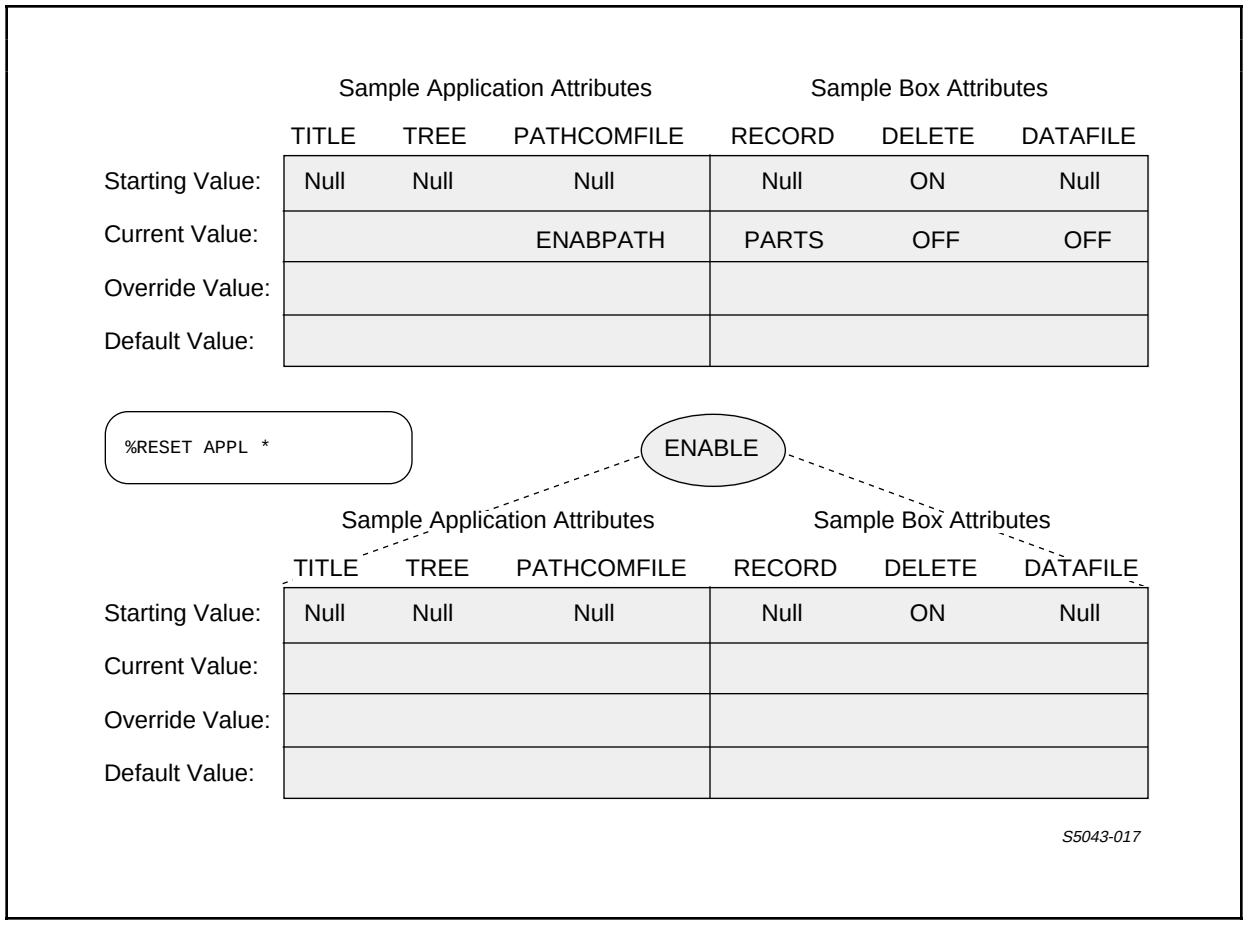


Figure 3-5. Resetting All Attributes with the RESET APPL * Command Option

Examples

The following example resets a box attribute:

```
SET BOX INSERT OFF
ADD BOX parts-1
ADD BOX supplier
RESET BOX INSERT  <--------------  resets the box attribute
ADD BOX order                      INSERT to ON, its starting
                                   value
```

The following example resets both box and application attributes:

```
SET BOX FILL ON
SET BOX DELETE OFF
SET BOX UPDATE OFF
SET BOX RECORD employee
ADD BOX employee
SET APPL SCOBOLOBJECT
SET APPL SCOBOLSOURCE newsrc
SET APPL PATHCOMFILE mypath
ADD APPL employee-prog
RESET APPL SCOBOLOBJECT, ABILITY  <-  resets the application
                                      attribute SCOBOLOBJECT
                                      to its starting value
                                      "POBJ"; also resets the
                                      box attributes DELETE,
                                      FILL, INSERT, and UPDATE
                                      to their starting
                                      values
```

SET Command

The SET command supplies a current value for an attribute. When you use the SET command, the value you supply remains current until you supply another value with a subsequent SET command, reset the attribute to its starting value with a RESET command, or exit from ENABLE. The syntax for the SET command is:

```
SET [ APPL ] <attribute> <value>
    [ BOX  ]

  [ , <attribute> <value> ] ...

  or

SET [ APPL ] LIKE <object> [ , <attribute> <value> ] ...
    [ BOX  ]
```

APPL

is a keyword that identifies application as the type of object for which attribute values are being set. If APPL is the default object type, you can omit this keyword when you set a value for an application attribute.

BOX

is a keyword that identifies box as the type of object for which attribute values are being set. To maintain compatibility with a previous version, ENABLE allows you to set a value for a box attribute when the default object type is either BOX or APPL.

<attribute>

is either an application attribute, or a box attribute. When BOX is the object type in effect you can specify only box attributes. When APPL is in effect you can specify both application and box attributes. Application and box attributes are classified as follows:

-->

Application Attributes:

PATHCOMFILE	SCOBOLOBJECT	TITLE	
PATHCOMSKELETON	SCOBOLSKELETON	TREE	
SCOBOLCOMPILER	SCOBOLSOURCE		
SCOBOLLIST	TERMINAL		

(You cannot specify application attributes when BOX is the object type in effect.)

Box Attributes:

BOXTITLE	EXCLUDE	INSERT	SERVERCLASS
CHECKDATA	FILL	NONSTOP	SIZE
DATAFILE	FLAG	READ	TMF
DELETE	HEADINGS	RECORD	UPDATE
DICTIONARY	INCLUDE	SCREENFORMAT	VALUES

Application and box attributes are described in Section 4.

<value>

is any legal value for the specified attribute.

LIKE <object>

sets the attribute values to those of the named object. The named object must appear in the object table, and be of the same type as that currently in effect.

Considerations

If you list the same attribute more than once in a single SET command, ENABLE issues an error message and does not set values for any of the attributes specified by that SET command.

Effect of a SET Command. A SET command supplies a current value for an attribute. To supply a current value for an application attribute, use the SET APPL command. To supply a current value for a box attribute, you can use either a SET BOX or a SET APPL command.

If you supply a current value for an application attribute, ENABLE uses that value to determine the description of an application added to the object table (unless you supply an override value for the same attribute when you enter an ADD command).

If you supply a current value for a box attribute, ENABLE uses that value to determine the description of a box added to the object table (unless you supply an override value for the same attribute when you enter an ADD command).

Resetting an Attribute. If you use the SET command to set a value for an attribute, you can use the RESET command to reset the attribute to its starting value.

The RESET APPL * command resets all box and application attributes to their starting values. The RESET BOX * command resets only the box attributes to their starting values.

Overriding a Value Set by a SET Command. If you use the SET command to supply a current value for an attribute, you can supply an override value for the same attribute by using the ADD command. If you supply an override value for an attribute with the ADD command, you do not change the current value of the attribute in the attribute table. The override attribute value applies only to the object you are adding. Refer to Section 2 for more information about override values.

Default Attribute Values. ENABLE supplies a default value for some box and application attributes if the starting, current, and override values of these attributes are null (blank). ENABLE only supplies these default values when adding an object (application or box) to the object table. ENABLE supplies default values for the following box and application attributes:

DATAFILE — ENABLE uses the name of the file identified in the record description as the default value of this attribute. If the record description does not identify a file name, ENABLE uses the record description name as the default value of this attribute.

TITLE — ENABLE uses <application-name> as the default value of the TITLE attribute.

TREE	ENABLE uses "(01 <application-name>-BOX)" as the default value of this attribute. ENABLE obtains this value by appending the four characters -BOX to the application name specified in the ADD APPL command.

Note that ENABLE uses the value of the TREE attribute to determine which box is associated with an application. If ENABLE supplies a default value for the TREE attribute, ENABLE checks to see if a box with the default name exists. If the box exists, ENABLE uses it for the application. If the box does not exist, ENABLE adds a box with the default name.

Application Attributes. Application attributes define characteristics that are global to an application, such as:

- The name of the PATHCOM command file
- The title that appears on the first line of the terminal screen
- The name and location of the skeleton files and SCREEN COBOL compiler

For a multifile application, you must set a value for the TREE attribute--the application attribute that associates added boxes with an application.

When you supply a value for an application attribute, you must include the keyword APPL if APPL is not the default object type. Table 4-1 summarizes the application attributes.

Box Attributes. Box attributes describe the characteristics of the box that represents a particular data base file. To provide a minimum description of a box, you must set a value for the RECORD attribute.

To maintain compatibility with a previous version of ENABLE, you can set a box attribute even if the default object type is APPL. This means that you can omit the object type parameter whenever you set a box attribute. Table 4-2 summarizes the box attributes.

Examples

The following example sets a value for the box attributes RECORD and DATAFILE:

```
SET BOX RECORD parts, DATAFILE part1
```

To set a value for the application attributes PATHCOMFILE and TITLE, you can enter:

```
SET APPL PATHCOMFILE mypath, TITLE "Entry Screen"
```

SHOW Command

The SHOW command displays the contents of the attribute table. The syntax of the SHOW command is:

```
SHOW [ APPL ] [ <attribute> ]
     [ BOX  ] [ ABIlITY     ]
              [ FORMAT      ]
              [ INTEGRITY   ]
              [ OTHER       ]
              [ *           ]
```

APPL

is a keyword that identifies application as the type of object whose attribute values are to be displayed.

BOX

is a keyword that identifies box as the type of object for which attribute values are to be displayed.

<attribute>

is either an application attribute, or a box attribute. When BOX is the object type in effect you can specify only box attributes. When APPL is in effect you can specify both application and box attributes. Application and box attributes are classified as follows:

Application Attributes:

PATHCOMFILE	SCOBOLOBJECT	TITLE
PATHCOMSKELETON	SCOBOLSKELETON	TREE
SCOBOLCOMPILER	SCOBOLSOURCE	
SCOBOLLIST	TERMINAL	

(You cannot specify application attributes when BOX is the object type in effect.)

-->

Box Attributes:

BOXTITLE	EXCLUDE	INSERT	SERVERCLASS
CHECKDATA	FILL	NONSTOP	SIZE
DATAFILE	FLAG	READ	TMF
DELETE	HEADINGS	RECORD	UPDATE
DICTIONARY	INCLUDE	SCREENFORMAT	VALUES

Application and box attributes are described in Section 4.

ABILITY

indicates that ENABLE is to display the current values of the attributes that control the ability of an application to delete, insert, read, or update a data base file. These attributes are: DELETE, FILL, INSERT, READ, and UPDATE.

FORMAT

indicates that ENABLE is to display the current values of the attributes that affect screen format. These attributes are: BOXTITLE 1, BOXTITLE 2, BOXTITLE 3, EXCLUDE, HEADINGS, INCLUDE, SCREENFORMAT, SIZE, and VALUES.

INTEGRITY

indicates that ENABLE is to display the current values of the attributes that affect the method that the application uses to ensure the integrity of a data base file. These attributes are: CHECKDATA, NONSTOP, and TMF.

OTHER

indicates that ENABLE is to display the current values of the following attributes: DATAFILE, DICTIONARY, FLAGS, RECORD, and SERVERCLASS.

-->

*
indicates that the current values of all attributes for the named or default object type are to be displayed.
If you do not specify an attribute or set of attributes, ENABLE displays current or default values for attributes of the object type in effect. Attributes with null values are omitted from display.

Considerations

The values displayed by the SHOW command are the values that ENABLE will use if you enter an ADD command. If SCOBOLOBJECT is set to null, the following message appears when you enter SHOW APPL SCOBOLOBJECT:

```
SCOBOLOBJECT   None:  no compilation
```

ENABLE also displays "None" if the current values of the TITLE or BOXTITLE attributes are null when you enter a SHOW command; for example:

The command:

```
SHOW APPL TITLE
```

causes ENABLE to display:

```
TITLE          (None)
```

The command:

```
SHOW BOX BOXTITLE 1
```

causes ENABLE to display:

```
BOXTITLE 1     (None)
```

Examples

To obtain the value of the application attribute TREE, enter:

```
SHOW APPL TREE
```

ENABLE returns:

```
TREE           ??
```

To obtain the value of the box attribute RECORD, enter:

```
SHOW BOX RECORD
```

To display the values of all application attributes including those for which you have supplied values, those for which ENABLE can supply default values, and those which have non-null starting values, enter:

```
SHOW APPL
```

To obtain the values of all the application attributes including those that retain non-null starting values but for which ENABLE cannot supply default values, enter:

```
SHOW APPL *
```

To obtain the values of all box attributes, enter:

```
SHOW BOX *
```

SECTION 4

ENABLE ATTRIBUTES

ENABLE attributes specify the characteristics of applications and boxes. Values are assigned to attributes using the commands described in Section 3.

Table 4-1 summarizes the attributes that describe applications. Table 4-2 summarizes the attributes that describe boxes. The syntax, function, and possible values for all ENABLE attributes are described in alphabetical order on the pages that follow.

Table 4-1. Summary of Application Attributes
(Continued next page)

Attribute	Characteristic	Starting Value
PATHCOMFILE	Identifies the name of the PATHCOM command file.	Null (No PATHCOM file is produced.)
PATHCOMSKELETON	Identifies the name of the PATHCOM skeleton file.	ENABPATS on the volume and subvolume where ENABLE runs.
SCOBOLCOMPILER	Identifies the SCREEN COBOL compiler that produces the SCREEN COBOL object code.	$system.system.scobolx
SCOBOLLIST	Identifies the name of a file to which the SCREEN COBOL listing is written.	Null (No SCREEN COBOL listing is produced.)
SCOBOLOBJECT	Identifies the name of the object files for the compiled SCREEN COBOL program.	POBJ on the current default volume and subvolume
SCOBOLSKELETON	Identifies the name of the SCREEN COBOL skeleton files.	ENABAPPS on the volume and subvolume where ENABLE runs
SCOBOLSOURCE	Identifies the name of the file to which the SCREEN COBOL source code is to be written.	Null (No SCREEN COBOL source file is produced.)

Table 4-1. Summary of Application Attributes (Continued)

Attribute	Characteristic	Starting Value
TERMINAL	Identifies the type of terminal on which the application will run.	The type of terminal from which you enter the ENABLE commands
TITLE	Identifies the text that is displayed on the first screen line.	Null (A default value is used when the application is added.)
TREE	Associates boxes with the application.	Null (A default value for a single-file application is supplied when the application is added.)

Table 4-2. Summary of Box Attributes (Continued next page)

Attribute Name	Function	Starting Value
Box Attributes That Affect Screen Format		
BOXTITLE 1 BOXTITLE 2 BOXTITLE 3	Identifies text that appears within a box on the screen.	Null (No text Null appears.) Null
EXCLUDE	Identifies fields from the record description that are not to appear in the box on the screen.	Null (All fields appear.)
HEADINGS	Identifes the set of labels to be used for field values on the screen.	DDLFIELDNAMES
INCLUDE	Identifies fields from the record description that are to appear in a box; also identifies the order in which the fields appear.	Null (Fields appear in same order as the record description.)
SCREENFORMAT	Identifies the choice of screen layout for a box.	UNCOMPRESSED
SIZE	Identifies the number of record occurrences that can be displayed in a box.	1

Table 4-2. Summary of Box Attributes (Continued)

Attribute Name	Function	Starting Value
Box Attributes That Affect Screen Format (Continued)		
VALUES	Indicates whether initial values from the record description appear within a box.	OFF
Box Attributes That Affect Capabilities		
DELETE	If set ON, indicates that the application can delete records from a file.	ON
FILL	If set ON, indicates that an automatic READ FIRST operation is performed.	OFF
INSERT	If set ON, the application can insert records in a file.	ON
READ	If set ON, the application can read records from a file.	ON
UPDATE	If set ON, the application can update records in a file.	ON

Table 4-2. Summary of Box Attributes (Continued)

Attribute Name	Function	Starting Value
Box Attributes That Affect Data Integrity		
CHECKDATA	If set ON, the application contains special code that verifies data read from numeric fields.	ON
NONSTOP	If set ON, the General Server runs as a NonStop process pair.	OFF
TMF	If set ON, indicates that a file is audited by the Transaction Monitoring Facility (TMF).	OFF

Table 4-2. Summary of Box Attributes (Continued)

Attribute Name	Function	Starting Value
	Other Box Attributes	
DATAFILE	Identifies the name of a data base file.	Null (A default value --from the DDL record description-- is supplied when the box is added.)
DICTIONARY	Identifies the location of the data dictionary that contains the record description identified by the value of the RECORD attribute.	The system, volume and subvolume in effect when you start ENABLE
FLAG 0 thru 99	Sets values for user flags defined in the SCREEN COBOL skeleton.	0
RECORD	Identifies the name of the DDL record description for a data base file.	Null (You must supply a value for this box attribute.)
SERVERCLASS	Provides a name for the server class to which the General Server belongs.	ENABLE-SERVER

BOXTITLE Attribute

The BOXTITLE attribute identifies up to three lines of text that are to appear within a box on the screen. The syntax of the BOXTITLE attribute is:

```
BOXTITLE { 1 } <string-literal>
         { 2 }
         { 3 }
```

BOXTITLE 1

  indicates that the associated <string-literal> is to appear on the first line of the box.

BOXTITLE 2

  indicates that the associated <string-literal> is to appear on the second line of the box.

BOXTITLE 3

  indicates that the associated <string-literal> is to appear on the third line of the box.

<string-literal>

  is the text that is to appear in the box. A <string-literal> must be enclosed in quotation marks (" "). The box will be at least as long as the length of <string-literal>. If <string-literal> is too long, ENABLE issues an error message when it generates an application that uses the box. Embedded quotation marks (") are not allowed.

The BOXTITLE attribute has neither a default nor a starting value.

Considerations

When you specify a value for one of the BOXTITLE attributes, the text that you specify appears left-justified within the box on the screen. If the box is continued across several pages of the screen, the text appears at the top of the box on each screen page.

The following list describes the relationship between the various BOXTITLE lines and the appearance of the box on the screen:

- If you set a value for BOXTITLE 1 but do not set a value for BOXTITLE 2 or BOXTITLE 3, the application displays the text on the first line of the box. A screen label and field for the box appears on the next box line.

- If you set a value for BOXTITLE 2 but do not set a value for BOXTITLE 1 or BOXTITLE 3, the application displays a blank line, the text, and the first line of screen labels or values.

- If you set a value for BOXTITLE 3 but do not set a value for BOXTITLE 1 or BOXTITLE 2, the application displays two blank lines, the text, and the first line of screen labels or values.

For a single-file application, you can set the BOXTITLE value to a string literal that is up to 79 characters long (78 for T16-651x terminals).

For multifile applications, you can supply a <string-literal> that is up to 79 characters long for the outermost box (78 for T16-651x terminals). You can supply a <string-literal> that is up to 71 characters long for a box at the second level of the tree structure (refer to the TREE attribute for information about levels of the tree structure). At each subsequent level of nesting, you can supply eight fewer characters for each <string-literal>.

Example

If you generate an application by including the following series of commands:

```
SET BOX BOXTITLE 1 "           Part Information"
SET BOX BOXTITLE 2 " "
SET BOX RECORD parts
ADD BOX parts-1
SET APPL TREE (01 parts-1)
SET APPL PATHCOMFILE newpath
ADD APPL parts-show
GENERATE APPL parts-show
```

the screen displayed by the application appears as shown in Figure 4-1.

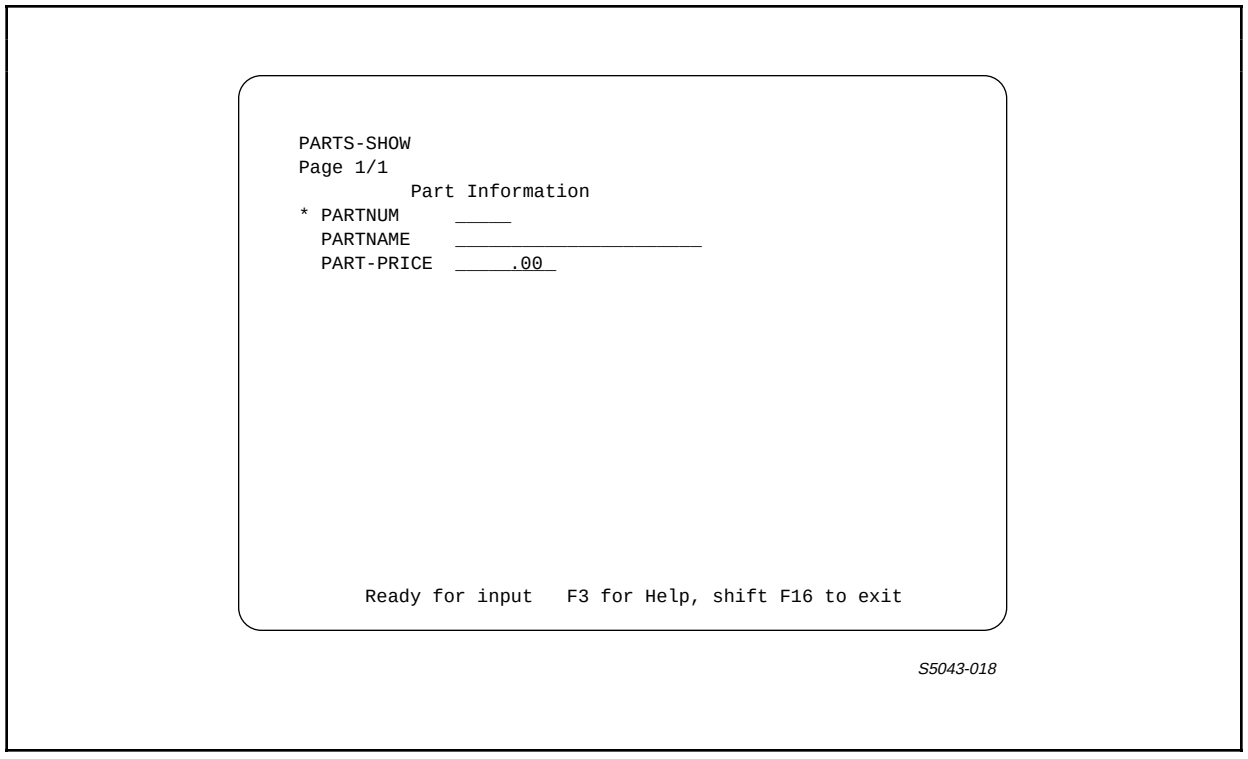


Figure 4-1. Sample Terminal Screen With BOXTITLE 1 and BOXTITLE 2 Values

CHECKDATA Attribute

The CHECKDATA attribute indicates whether the application does, or does not, contain special code that verifies the data type of data read from numeric fields in a data base file. The syntax of the CHECKDATA attribute is:

```
CHECKDATA { ON  }
          { OFF }

ON

  indicates that the portion of the SCREEN COBOL requester
  program that applies to the box checks the contents of
  numeric fields.

OFF

  indicates that the portion of the SCREEN COBOL requester
  program that applies to the box does not check the
  contents of numeric fields.

The starting value of the CHECKDATA attribute is ON.
```

Considerations

When CHECKDATA is ON and the application reads invalid numeric data (a non-numeric value in a numeric field) from a data base file, the application issues a warning message. It then replaces the invalid data with zeros for display purposes only. The application does not change the actual data in the data base file unless you request an UPDATE operation.

If CHECKDATA is OFF and the application reads invalid data from a numeric field, the terminal will abort when the application tries to display the invalid data.

Example

The following command sets the value of the box attribute CHECKDATA to OFF:

```
SET BOX CHECKDATA OFF
```

DATAFILE Attribute

You can use the DATAFILE attribute to identify the data base file that a box represents. The syntax of the DATAFILE attribute is:

```
DATAFILE <data-file-name>

<data-file-name>

  is the name of the file that the box represents and must
  have the following form:

  [\<system>.][$<volume>.][<subvolume>.]<file-name>

    \<system>     is the name of the system where the file
                  resides.

    $<volume>     is the name of the volume where the file
                  resides.

    <subvolume>   is the name of the subvolume where the
                  file resides.

    <file-name>   is a valid Tandem disc file name.

  If you omit <system>, <volume>, or <subvolume>, ENABLE
  uses the CMDSYS and CMDVOL default values displayed when
  you enter the operating command ENV.

  This name must conform to the Tandem file name
  conventions described in Section 5 under the heading
  "File Name Expansion."

The starting value of the DATAFILE attribute is null.
ENABLE supplies a default value for this attribute
based on the value of the RECORD attribute.  ENABLE
checks the current dictionary for a description of the
record, and uses the file named in that description as a
default value for DATAFILE.  If the record description
names no file, ENABLE uses the record name as the default
value.
```

Consideration

ENABLE uses the value of the DATAFILE attribute when generating the PATHCOM command file. When you set a value for DATAFILE, you preclude any file name identified in the record description from being used.

Examples

Suppose that the following record description identifies more than one data base file:

```
RECORD manager.
  FILE IS "employ" KEY-SEQUENCED.
    02 empnum      PIC 9(4).
    02 empname     PIC 9(18).
    02 dept-id     PIC 9(4).
  KEY 0 IS empnum.
  KEY "di" IS dept-id.
END
```

With this record description, the default value of the DATAFILE attribute is "employ."

You can use the following commands to associate a box with the file "employ":

```
SET BOX RECORD manager
ADD BOX manager
```

To describe other files with the same record description, a value for the DATAFILE box attribute must be set explicitly.

The following commands can be used to associate a box with the "manager" record description and the file "employ1":

```
SET BOX RECORD manager
SET BOX DATAFILE employ1
ADD BOX employees
```

DELETE Attribute

The DELETE attribute indicates whether the application can, or cannot, delete records from the file that a box represents. The syntax for the DELETE attribute is:

```
DELETE { ON  }
       { OFF }
```

ON

indicates that the application can delete records from the data base file.

OFF

indicates that the application cannot delete records from the data base file.

The starting value of the DELETE attribute is ON.

Considerations

If the value of DELETE is ON, the value of the READ attribute must also be ON. ENABLE imposes this restriction because an application must read a record before that record can be deleted.

An application cannot delete records from entry-sequenced or unstructured files. If DELETE is set to ON for a box that represents such a file, ENABLE sets DELETE to OFF when the box is added, and issues a warning message.

Example

Consider the value of the DELETE attribute in the following series of ENABLE commands:

```
SET BOX DELETE OFF  <-----  sets DELETE to OFF
SET BOX RECORD parts
ADD BOX parts-detail
```

An application that uses the "parts-detail" box cannot delete records from the file that the box represents.

DICTIONARY Attribute

The DICTIONARY attribute identifies the location of the data dictionary from which ENABLE is to obtain a record description. The syntax of the DICTIONARY attribute is:

```
DICTIONARY { <subvolume>                          }
           { $<volume>.<subvolume>                }
           { \<system>.$<volume>.<subvolume> }
```

<subvolume>

identifies the subvolume where the dictionary resides.

$<volume>

identifies the volume where the dictionary resides.

\<system>

identifies the system where the dictionary resides.

This name must conform to the Tandem file name conventions described in Section 5 under the heading: "File Name Expansion."

The starting value of the DICTIONARY attribute is the system, volume, and subvolume in effect when you start ENABLE.

Consideration

ENABLE can read record descriptions from several different dictionaries within a session. ENABLE reads the record description from the dictionary identified by the value of the DICTIONARY attribute at the time a box is added.

Examples

The following commands obtain record descriptions from two different dictionaries within an ENABLE session.

```
SET BOX DICTIONARY $mkt.sample1
SET BOX RECORD parts
ADD BOX part-1

SET BOX DICTIONARY $mkt.sample2
SET BOX RECORD order
ADD BOX order-box
```

EXCLUDE Attribute

The EXCLUDE attribute identifies fields from a record description to be excluded from display within a box. The syntax of the EXCLUDE attribute is:

```
EXCLUDE { <qualified-field-name>                }
        { ( <qualified-field-name> [ , ... ] ) }

<qualified-field-name>

  is the name of a field to be excluded. If you exclude
  more than one <qualified-field-name>, you must enclose
  the field names in parentheses. You can continue the
  EXCLUDE attribute on to subsequent command lines. To
  continue processing interactively, you must enter the
  closing parenthesis, even if you encounter an error.

  To avoid ambiguity, a field name specified as a value for
  the EXCLUDE attribute must be unique within the record
  description associated with a box. If
  <qualified-field-name> is either the primary key, or a
  courtesy key (Record Number field) of a record, use the
  keyword "KEY" to exclude this field. If
  <qualified-field-name> is neither unique nor a key field,
  you must qualify it by including a dominant group name.
  Refer to Section 2 for more information about field name
  qualification.

The EXCLUDE attribute has neither a starting value nor a
default value. If you do not supply a value for either
the EXCLUDE or INCLUDE attribute, all fields from a record
description are displayed within the box (with the
exception of the join field of a nested box), in the same
order that they appear in the record description.
```

Considerations

When you supply a value for the EXCLUDE attribute, the generated application does not display screen labels or data values for the excluded fields on the screen.

ENABLE cannot use a box for an application if you exclude:

- All of the fields described in a record description. If you try to exclude all fields, ENABLE issues an error message stating that "All fields have been excluded from this box."

- For any box except the box at the first level of the tree structure, all fields except the join field. If you exclude all fields except the join field from a such a box, ENABLE issues the error message: "No field is left to be displayed in box." Refer to the description of the TREE attribute for an explanation of the levels of a tree structure.

- The primary key field of a key-sequenced file or a unique alternate key field from any type of file when INSERT is set ON. ENABLE cannot generate the application because an insert operation could cause a duplicate key error. If you try to exclude a unique key field, ENABLE issues the error message: "Field corresponding to unique key has been excluded - <field-name>."

- All key fields from a box. If you exclude all key fields from a box, ENABLE issues the error message: "All key fields have been excluded from this box."

- The join field from a box. Refer to the description of the TREE attribute for an explanation of a join field.

- Any component of a group join field. All components of a group used as a join field must be in the same order as they appear in the dictionary, and all fields in the group must be present.

Excluding an Elementary Item From a Group Key. If a group item is an alternate key field and you exclude the first elementary item from that group, the application will not use the group item as an alternate key. For example, consider the following record description:

```
RECORD employee.
FILE IS ...  KEY-SEQUENCED.
02 empnum       PIC 9(4).
...
02 dept.  <--------------------  group
  04 regnum     PIC 9(4).  <---  first elementary item in the
  04 branchnum  PIC 9(4).        group
...
KEY 0 IS empnum.
KEY "dp" IS dept.
END
```

If you exclude "regnum," a generated application does not recognize "dept" as an alternate key; therefore, the application cannot read records for the file by using this alternate key field.

If a group item is a key field and you exclude any item other than the first item in the group, the application uses only the first intact item of the group as a key field. Consider, for example, the following record description:

```
RECORD abc.
FILE IS ... .
02 A.
  03 AA.
   04 AAA ... .
   04 AAB ... .
  03 AB   ... .
  03 AC.
   04 ACA ... .
   04 ACB ... .
...
KEY 0 IS A.
END
```

In this record description, the group A is a key field. Normally, the application would display this field as follows:

```
* A
    AA
      AAA
      AAB
    AB
    AC
      ACA
      ACB
```

where the asterisk (*) identifies A as a primary key field. In this case, the application can use the entire group as a key field. If you exclude ACB, for example, the application will display these fields as follows:

```
  A
* AA
    AAA
    AAB
  AB
  AC
    ACA
```

In this case, the application uses AA as a key field.

If you exclude all of the elementary items of a group field, the group field does not appear on the screen. For example, suppose that you exclude the elementary items of the following group:

```
02 address.
  04 street  PIC X(20).
  04 city    PIC X(20).
  04 state   PIC X(20).
```

by entering:

```
SET EXCLUDE (street, city, state)
```

The application will not display a screen label named "address" because you have excluded all of the elementary fields from the "address" group.

Examples

The following example excludes a single field:

```
SET BOX EXCLUDE price
```

The following example excludes several fields:

```
SET BOX EXCLUDE (street, city, state, zipcode,
                 telephone-number)
```

FILL Attribute

The FILL attribute indicates whether the application does, or does not, automatically read a record (or records) for a box under the conditions described below. The syntax of the FILL attribute is:

```
FILL { ON  }
     { OFF }
```

ON

indicates that, under certain circumstances, the application is to perform an automatic read operation on the file that the box represents.

OFF

indicates that the application is not to perform an automatic read operation under any circumstances.

The starting value of the FILL attribute is OFF.

Considerations

When you supply ON as a value for the FILL attribute, a generated application can perform an automatic read operation on a data base file. The conditions under which the application performs the read operation depends upon the level at which the box resides in the tree structure of the application.

For a box at the highest level of the tree structure (the outermost box), two cases are possible:

- If the box has OFF as the value of the VALUES attribute, the application, upon execution, automatically reads the first record or records (if any) from the data base file.

- If the box has ON as the value of the VALUES attribute, the application, upon execution, automatically reads a record or records from the data base file by using the initial values in the record description as a starting point. If there are no records in the file, the application displays the initial values on the screen.

For a box at a subsequent level of the tree, the application performs an automatic read operation whenever a user performs an operation other than DELETE (INSERT, READ, or UPDATE) on the higher level box to which that box is linked.

Suppose, for example, that you generate an application with the following value for the TREE attribute:

```
TREE ( 03 parts
          05 supplier LINK parts.partno
          TO OPTIONAL supplier.partnum )
```

For this application, the "parts" box is at the highest level of the tree structure. If FILL is ON and VALUES is OFF for the "parts" box, the application automatically reads the first record in the file associated with this box.

If FILL is ON for the "supplier" box, which is at the next level of the tree, the application automatically performs a read operation on the file associated with this box whenever an operation (other than DELETE) is performed on the "parts" box.

Control Boxes. If a record description contains a DDL VALUE clause, you can use the box that represents this file as a control box. To do this, supply ON for both the FILL and the VALUES attributes, and identify the box as being at the highest level of the tree structure (the outermost box). For example, consider the the following partial record descriptions:

```
RECORD region.
FILE IS region KEY-SEQUENCED.
02 regnum   PIC 9(4) VALUE IS 4.
02 regname  PIC X(12) VALUE IS "SOUTHEAST  ".
...
KEY 0 IS regnum.
END

RECORD branch.
FILE IS branch RELATIVE
02 branchnum   PIC 9(4).
02 regnum      PIC 9(4).
02 branchname  PIC X(15).
...
KEY 0 IS branchnum.
KEY "rg" is regnum.
END
```

If you use the following commands to generate the application:

```
SET BOX FILL ON
SET BOX DELETE OFF, INSERT OFF, UPDATE OFF, READ OFF
SET BOX VALUES ON
SET BOX RECORD region
ADD BOX region-box
RESET BOX *
SET BOX RECORD branch
ADD BOX branch-box
SET APPL PATHCOMFILE pf1
SET APPL TREE ( 01 region-box
                   02 branch-box LINK region-box
                   TO OPTIONAL branch-box VIA regnum )
ADD APPL region-4
GENERATE APPL region-4
```

the application, upon execution, performs an initial read operation on the file represented by "region-box." If that file contains a record for region 4, the application retrieves that record. Because READ is set to OFF, the record for region 4 is the only record that the application can access. Because the application can only read one record for "region-box," it can only read records for "branch-box" if their join field value is 4. The application is limited to this subset of branch records because:

- The application must either insert or read a record from "region-box" before it can perform any operations on "branch-box" (It cannot insert a record for "region-box" because INSERT is set to OFF for that box.)
- The application uses the value of the "regnum" as a key to read or insert a record for "branch-box"
- A "regnum" value of 4 is the only value that the application can read for "region-box"

Example

The following command sets the box attribute FILL to ON:

```
SET FILL ON
```

FLAG Attribute

You can use the FLAG attribute to set values for flags that can be referred to in the application skeleton. You can modify the application skeleton through special commands that ENABLE uses to test and act upon the value of a user flag. If you modify the application skeleton by associating user flags with a block of SCREEN COBOL code, ENABLE uses the value of the FLAG attribute to determine which blocks of code are to be included or omitted from the generated SCREEN COBOL program. The syntax of the FLAG attribute is:

```
FLAG { <flag-number> } <flag-value>
     { *             }

<flag-number>

  is a positive integer from 0 to 99.  ENABLE uses the
  value of <flag-number> to identify a specific user flag
  in the application skeleton.  To determine the
  appropriate value for <flag-number>, subtract 100 from
  the value of the user flag number in the skeleton.

*

  indicates that all user flags are set to <flag-value>.

<flag-value>

  is a positive integer from 0 to 255.  ENABLE uses
  <flag-value> to determine which blocks of code are to be
  generated into or omitted from the portion of the
  application that refers to the box.

The starting value of all FLAG attributes is 0.
```

Consideration

You can specify user flags so that ENABLE includes an entire block of code in the application, includes only certain lines of the block of code in the application, or omits the block of code entirely. Refer to Appendix D for more information about user flags.

Examples

The following command sets user flag number 1 to 79:

```
SET BOX FLAG 1 79
```

In the skeleton program, user flags have values from 101 to 200. The preceding command references flag number 101 in the skeleton program.

The following FLAG command sets all of the user flags associated with a box to 1:

```
SET BOX FLAG * 1
```

HEADINGS Attribute

The HEADINGS attribute indicates the source of any screen-field labels that are to appear within the box. If screen-field labels are to appear, the value of this attribute indicates whether headings or field names from the record description are used as screen labels. The syntax of the HEADINGS attribute is:

```
HEADINGS { DDLFIELDNAMES  }
         { DDLHEADINGS    }
         { NULL           }
```

DDLFIELDNAMES

indicates that the application uses field names from the record description as screen labels.

DDLHEADINGS

indicates that the application uses any headings specified in the DDL record description as screen labels.

NULL

indicates that no screen labels are displayed for fields within the box.

The starting value of the HEADINGS attribute is DDLFIELDNAMES.

Consideration

If you set HEADINGS to DDLHEADINGS and the record description does not identify a heading for a field, ENABLE uses the field name as the screen label.

For compatibility with previous versions, ENABLE also supports the following values for the HEADINGS attribute:

ON is synonymous with DDLHEADINGS

OFF is synonymous with DDLFIELDNAMES

Example

If you generate an application by including the following commands:

```
SET BOX RECORD parts
SET BOX HEADINGS DDLHEADINGS  <-  sets HEADINGS to
ADD BOX parts-1                   DDLHEADINGS
SET APPL HEADINGS NULL  <-------  sets HEADINGS to NULL
SET APPL SCREENFORMAT COMPRESSED
SET BOX RECORD supplier
ADD BOX suppliers
RESET BOX HEADINGS  <-----------  resets HEADINGS to
...                               DDLFIELDNAMES
```

The application uses headings from the "parts" record description as screen labels for the "parts-1" box. The application does not display any labels for the "suppliers" box.

INCLUDE Attribute

The INCLUDE attribute identifies fields from a record description that the application is to display within a box. This attribute also specifies the order in which the fields are to appear.

Fields appear in the same order that you list them, and any field that you do not explicitly list with this attribute is omitted from the display. The syntax of the INCLUDE attribute is:

```
INCLUDE {  <qualified-field-name>                }
        {  ( <qualified-field-name> [ , ... ] ) }

<qualified-field-name>

  is the unique name of a field in the record description.
  If you include more than one field, you must enclose the
  field names in parentheses.  When you use parentheses,
  you can continue the INCLUDE attribute on subsequent
  command lines.

  To avoid ambiguity, the field name must be unique within
  the record description associated with the box.  If the
  field name is not unique, you must qualify it by
  including the name of a unique dominant group.  Refer to
  Section 2 for information about qualifying field names.
  If <qualified-field-name> is either the primary key, or a
  courtesy key (Record Number field) of a record, use the
  keyword "KEY" to include this field.

The INCLUDE attribute has neither a starting value nor a
default value.  If you do not supply a value for either the
EXCLUDE or INCLUDE attribute, all fields from a record
description are displayed within the box in the same order
that they appear in the record description.
```

Considerations

When you supply a value for the INCLUDE attribute:

- If you include more than one field, the application displays the screen label and field pairs for the fields in the order in which you enter the field names in the SET INCLUDE command.

- You must include any join fields for a box. (Refer to the TREE attribute for an explanation of a join field.) If a join field is a group field, you cannot reorder the items that make up that group.

- If the value of the INSERT attribute is ON, you must include the primary key of a key-sequenced file or a unique alternate key for any file type.

Since ENABLE excludes any field that you do not list with the INCLUDE attribute, you must include any field that cannot be excluded when you supply a value for the EXCLUDE attribute. Refer to the discussion of the SET EXCLUDE command for more information about these fields.

Reordering the Elementary Items of a Group. If you use the INCLUDE attribute to reorder the elementary items that make up a composite key (a group field that is a primary or alternate key field), the application cannot use the entire group item as a key field. Instead, the application uses the first field (elementary or group) that has not been reordered as a key.

If you do not include the first elementary field as part of the composite key, the application does not use any part of the composite key as a key. The application cannot use the group as a key field because the user of the application would have no way of knowing whether an element is the first, second, or third item of the key field. (In other words, a READ GENERIC operation could not be defined.)

Consider, for example, the following record description:

```
RECORD zip.
FILE IS ... KEY-SEQUENCED.
02 aaa   PIC X.
02 abc.
   04 a  PIC X(4).
   04 b  PIC X(4).
   04 c  PIC X(4).
02 d.
  04 ee  PIC 9(4).
  04 nn  PIC 9(4).
KEY 0 IS aaa.
KEY "ab" IS abc.
END
```

In the "zip" record description, "abc" is a composite alternate key field made up of the elementary fields "a," "b," and "c." If you add a box with the following commands:

```
SET BOX RECORD zip
SET BOX INCLUDE (aaa,c, a, b)
ADD BOX zipper
```

the fields for box "zipper" will appear on the screen displayed by the application as:

```
* AAA   _
  ABC
    C   ____
+   A   ____
    B   ____
```

Notice that a plus sign appears before the screen label for field "A." This sign identifies the field that the application can use as an alternate key. The application can use "A" as a key because it is the first field of the composite key "ABC."

If the first field of a composite key is a group field and you reorder that group field, the application uses the first elementary field of that group as a key.

For example, consider the following record description:

```
RECORD zot.
FILE IS zot ...
02 ABC.
  04 AA.
    06 BBB    PIC 9.
    06 CCC    PIC 9.
  04 BB
    06 DDD    PIC X.
    06 EEE    PIC X.
...
KEY "ac" IS ABC.
END
```

In the "zot" record description, "abc" is a composite alternate key field consisting of the following group fields: "aa" and "bb." If you generate an application by including the following commands:

```
SET BOX RECORD zot
SET BOX INCLUDE (ddd, bbb, ccc, eee)
ADD BOX zing
```

you reorder the elementary fields of both "aa" and "bb." Because these fields are reordered, an application that uses box "zing" displays these fields as follows:

```
 ABC BB
     DDD  _
+  AA
     BBB  _
     CCC  _
   BB
   EEE   _
```

Notice that the application can use field "bbb," the first elementary field of the composite key "ABC," as an alternate key field. After you add the box for which the value of the INCLUDE attribute applies, you should use the RESET command to reset INCLUDE to its starting value.

Examples

The following example of the INCLUDE attribute command includes fields named "partno," "partname," and "price":

```
SET BOX INCLUDE (partno, partname, price)
```

To display only one field from a record description, you can enter:

```
SET BOX INCLUDE partno
```

INSERT Attribute

The INSERT attribute indicates whether the application can, or cannot, add a record to the data base file represented by a box. The syntax of the INSERT attribute is:

```
INSERT { ON  }
       { OFF }
```

ON

indicates that the application can add records to a data base file.

OFF

indicates that the application cannot add records to a data base file.

The starting value of the INSERT attribute is ON.

Consideration

For multifile applications, the value of INSERT must be OFF for any child box that represents an entry-sequenced or unstructured file where the join field for that box is the courtesy key (Record Number field).

ENABLE imposes this restriction because the application ignores the record number field when inserting records in files with these file types.

Example

The following command sets the INSERT attribute to OFF:

```
SET INSERT OFF
```

NONSTOP Attribute

The NONSTOP attribute indicates whether the General Server is, or is not, to run as a NonStop process pair when accessing the data base file represented by a box. The syntax of the NONSTOP attribute is:

```
NONSTOP { ON  }
        { OFF }
```

ON

indicates that the General Server is to run as a NonStop process pair.

OFF

indicates that the General Server is not to run as a NonStop process pair.

The starting value of the NONSTOP attribute is OFF.

Considerations

A given application can use different server classes to access different data base files. If you are generating a multifile application, the value of NONSTOP can be ON for one box and OFF for another, as long as the same server class of the General Server does not access files represented by both boxes.

ENABLE does not allow you to add a box if the value of both TMF and NONSTOP is ON.

Example

If you generate an application by including the following commands:

```
SET BOX NONSTOP ON
SET BOX SERVERCLASS e-nonstop
SET BOX RECORD parts
ADD BOX parts-1
BOX RECORD orders
ADD BOX orders
RESET BOX NONSTOP
SET BOX SERVERCLASS tmf-serve
SET BOX TMF ON
SET BOX RECORD supplier
ADD BOX supplier-1
RESET BOX *
```

a copy of the General Server identified by the name "e-nonstop" accesses the files represented by the "parts-1" and "orders" boxes. This copy of the General Server will run as a NonStop process pair. Another copy of the General Server belonging to the server class "tmf-serve" accesses the file represented by the "supplier-1" box. The file represented by this box is audited by TMF.

PATHCOMFILE Attribute

The PATHCOMFILE attribute identifies the file to which ENABLE writes the set of PATHCOM commands it produces. These PATHCOM commands are used to configure a PATHWAY system under which the application runs. The syntax of the PATHCOMFILE attribute is:

```
PATHCOMFILE <file-name> [!]

  NOTE:  APPL must be the object type in effect when you
  supply a value for this application attribute.

<file-name>

  identifies the name of the file to which ENABLE is to
  write the generated PATHCOM commands.  ENABLE writes
  <file-name> in edit-type file format.

!

  indicates that an existing file is to be overwritten.  If
  an edit-type file named <file-name> already exists and
  you enter the exclamation point symbol, ENABLE writes
  over the file when a GENERATE command is executed for an
  application with this value of the PATHCOMFILE attribute.

The PATHCOMFILE attribute has neither a starting nor
default value.
```

Considerations

If you do not supply a value for the PATHCOMFILE attribute, ENABLE does not generate a PATHCOM command file.

If you do not generate the SCREEN COBOL object code (see the SCOBOLOBJECT attribute), you will have to edit the PATHCOM command file before you use it to configure a PATHWAY system. In this case ENABLE issues a warning message when it generates the application.

ENABLE issues an error message if:

- A file named <file-name> already exists but you do not specify the ! symbol.
- You specify the ! symbol but <file-name> is not an edit-type file.

Example

The following example identifies the file "enabpath" as the destination file to which ENABLE is to write the PATHCOM commands:

```
SET APPL PATHCOMFILE enabpath !
```

PATHCOMSKELETON Attribute

The PATHCOMSKELETON attribute identifies the file that ENABLE is to use as the PATHCOM skeleton. (ENABLE uses this file when generating the PATHCOM command file.) The syntax of the PATHCOMSKELETON attribute is:

```
PATHCOMSKELETON <skeleton-file-name>

  NOTE:  APPL must be the object type in effect when you
  supply a value for this application attribute.

<skeleton-file-name>

  identifies the name of the file containing the PATHCOM
  skeleton.

The starting value for the PATHCOMSKELETON attribute is
ENABPATS on the volume and subvolume where ENABLE runs.
```

Considerations

Although the PATHCOM skeleton supplied with ENABLE is suitable for most applications, you can modify the PATHCOM skeleton before ENABLE uses it to generate an application. To modify the skeleton, do the following:

1. Use either the File Utility Program (FUP) or the editor to make a copy of the PATHCOM skeleton file supplied with ENABLE.

2. Use the editor to make the desired changes to the copy.

To use the edited version of the PATHCOM skeleton file when you generate an application, supply the filename of the edited version as the value of the PATHCOMFILE attribute.

Example

The following example identifies "$mkt.sample.pathskel" as the name of the file to be used as the PATHCOM skeleton:

```
SET APPL PATHCOMSKELETON $mkt.sample.pathskel
```

READ Attribute

The READ attribute indicates whether the application can read a record or records from the data base file represented by the box. The syntax for the READ attribute is:

```
READ { ON  }
     { OFF }

ON

  indicates that the application can read a record or
  records from the data base file.

OFF

  indicates that the application cannot read a record or
  records from the data base file.

The starting value of the READ attribute is ON.
```

Considerations

You can generate an application that cannot read any records for a box. You do this if you supply OFF as a value for both READ and FILL. A box of this sort may be used for inserting records only.

If the value of either UPDATE or DELETE is ON, then the value of READ must also be ON.

Example

An application generated with the following commands cannot read records for the "parts-1" box, but can read records for the "supplier-1" and "order" boxes:

```
SET BOX DELETE OFF, UPDATE OFF
SET BOX READ OFF  <-----------  sets READ to OFF
SET RECORD parts
ADD BOX parts-1
RESET BOX READ  <-------------  resets READ to ON
SET BOX RECORD supplier
ADD BOX supplier-1
SET BOX RECORD orders
ADD BOX order
```

RECORD Attribute

The RECORD attribute identifies the name of the record description used for the box. ENABLE uses the record description to determine the record, fields, and structure of the data base file represented by a box. The syntax of the RECORD attribute is:

```
RECORD <external-record-name>

<external-record-name>

  is the name of the record description.
  <external-record-name> must be a record name found in the
  dictionary specified by the value of the DICTIONARY
  attribute.

You must set a value for the RECORD attribute.
```

Considerations

When you set RECORD to the name of a record description, ENABLE does not verify that the record description exists until you attempt to ADD the box to the object table.

To maintain compatiblility with previous versions of ENABLE, when generating a single-file application the RECORD attribute is the only attribute for which you must supply a value. You do not need to add a box or set a value for the TREE attribute in this case.

DDL Record-Description Limitations. Because ENABLE uses record descriptions to generate SCREEN COBOL source code, the syntax of the DDL RECORD statement must be compatible with SCREEN COBOL representation capabilities. The limitations involved are described in detail in Section 1, under "Dictionary Files."

Example

The following command identifies a record description named "parts":

```
SET BOX RECORD parts
```

SCOBOLCOMPILER Attribute

The SCOBOLCOMPILER attribute identifies the SCREEN COBOL compiler that produces object code for an application. The syntax of the SCOBOLCOMPILER attribute is:

```
SCOBOLCOMPILER [ <compiler-name> ]

  NOTE:  APPL must be the object type in effect when you
  supply a value for this application attribute.

<compiler-name>

  identifies the file containing the SCREEN COBOL compiler.

  If you omit <compiler-name> with the SCOBOLCOMPILER
  attribute, ENABLE suppresses compilation of the SCREEN
  COBOL program.

The starting value of the SCOBOLCOMPILER attribute is
"$system.system.scobolx."
```

Considerations

To obtain SCREEN COBOL source code before compiling the application, you can omit the <compiler-name> and set a value for the SCOBOLSOURCE attribute. If you do this, you must compile the SCREEN COBOL source code and edit the generated PATHCOM file before you execute the application.

You can use this attribute to specify an alternate compiler for load balancing.

If you attempt to suppress SCREEN COBOL compilation and request a listing using the SET SCOBOLLIST command, ENABLE responds with an error message.

Example

The following command specifies the SCREEN COBOL compiler "$mkt.sample.scobolx."

```
SET APPL SCOBOLCOMPILER $mkt.sample.scobolx
```

SCOBOLLIST Attribute

The SCOBOLLIST attribute identifies the file to which ENABLE directs a compilation listing of the SCREEN COBOL program. The syntax of the SCOBOLLIST attribute is:

```
SCOBOLLIST [ <file-name> [!] ]

  NOTE:  APPL must be the object type in effect when you
  supply a value for this application attribute.

<file-name>

  identifies the file to which the SCREEN COBOL compiler is
  to write the compilation listing.  <file-name> can be a
  terminal, line printer, magnetic tape unit, process (for
  example, a spooler process), or an unstructured disc
  file.  If <file-name> is a disc file, you must specify an
  extent size for the file that is large enough to
  accommodate the listing output when you create the file.
  Note that the default extent size of 1 is insufficient.

  If <file-name> does not exist, you must include the
  exclamation point symbol.

  If you omit the <file-name> option but enter the
  SCOBOLLIST attribute, the SCREEN COBOL compiler directs
  the compilation listing to the current ENABLE list file.

!

  indicates that the SCREEN COBOL compiler is to overwrite
  an existing file.  If a file named <file-name> already
  exists and you include !, the data in the file is purged
  before the write operation takes place.  If a file named
  <file-name> does not exist and you include !, ENABLE
  creates the file.  If a file named <file-name> exists and
  you omit the !, ENABLE issues an error message.

If you do not set a value for the SCOBOLLIST attribute, the
SCREEN COBOL compiler writes the compilation listing to a
temporary file with a name of the form ZZENLnnn.  If no
errors occur during compilation, ENABLE purges this file.
```

Example

The following command identifies "$mkt.sample.list" as the file to which the SCREEN COBOL compilation listing is to be directed:

```
SET APPL SCOBOLLIST $mkt.sample.list
```

SCOBOLOBJECT Attribute

The SCOBOLOBJECT attribute identifies the object file to which the SCREEN COBOL compiler directs the compiled SCREEN COBOL program. The syntax of the SCOBOLOBJECT attribute is:

```
SCOBOLOBJECT [ <file-name> ]

  NOTE:  APPL must be the object type in effect when you
  supply a value for this application attribute.

<file-name>

  identifies a name that the SCREEN COBOL compiler uses for
  the files containing the SCREEN COBOL object code and the
  directory for this object code.  <file-name> must conform
  to the SCREEN COBOL compiler requirements and must not
  exceed five characters.

  If you omit <file-name> with the  SCOBOLOBJECT attribute,
  ENABLE suppresses compilation of the SCREEN COBOL
  program.

The starting value of the SCOBOLOBJECT attribute is POBJ on
the default volume and subvolume.
```

The SCREEN COBOL compiler writes the object code and directory information on two disc files. The compiler appends the characters COD to <file-name> to determine the name of the file that is to contain the object code. The compiler appends the characters DIR to <file-name> to determine the name of the file that is to contain the directory for the object code.

If files with these names exist, the SCREEN COBOL compiler appends the new object code to these files.

If the files already exist and contain object code for an application with the same program-unit name, the SCREEN COBOL compiler adds the new version of the object code to the object files. The old version of the code remains in the object files; however, PATHWAY uses the newest version of this code. To remove unwanted old versions of object code, use the SCREEN COBOL Utility Program (SCUP).

Refer to the ENABLE User's Guide for a discussion of removing old versions of object code.

Consideration

If you want to obtain SCREEN COBOL source code before using the application, you can set a value for SCOBOLSOURCE and enter the SCOBOLOBJECT attribute without a value for the <file-name> parameter. If you do this, you must compile the SCREEN COBOL source code and edit the generated PATHCOM file before you execute the application.

Example

The following command identifies "$mkt.sample.nobj" as the names of the files to which the SCREEN COBOL object code is to be directed:

```
SET APPL SCOBOLOBJECT $mkt.sample.nobj
```

To suppress compilation of the SCREEN COBOL source code, enter:

```
SET APPL SCOBOLOBJECT
```

SCOBOLSKELETON Attribute

The SCOBOLSKELETON attribute identifies the file that ENABLE is to use as the SCREEN COBOL skeleton. The syntax of the SCOBOLSKELETON attribute is:

```
SCOBOLSKELETON <file-name>

  NOTE:  APPL must be the object type in effect when you
  supply a value for this application attribute.

<file-name>

  is the name of a SCREEN COBOL application skeleton.

The starting value of the SCOBOLSKELETON attribute is
ENABAPPS on the volume and subvolume where ENABLE runs.
```

Considerations

ENABLE uses the SCREEN COBOL skeleton to produce the application.

You can create a copy of the skeleton with FUP, and modify it using the editor. You can then supply the name of this modified version to ENABLE using the SCOBOLSKELETON attribute.

Refer to Appendix D for more information about creating a modified version of the SCREEN COBOL skeleton.

If you modify the original SCREEN COBOL skeleton file, you do so AT YOUR OWN RISK. If you change the name of a modified version to the default skeleton file name, you do so AT YOUR OWN RISK.

Example

The following command identifies "$mkt.sample.scobskel" as the name of the file that ENABLE is to use as the SCREEN COBOL skeleton file:

```
SET APPL SCOBOLSKELETON $mkt.sample.scobskel
```

SCOBOLSOURCE Attribute

The SCOBOLSOURCE attribute identifies a file to which ENABLE writes the SCREEN COBOL source code. The syntax for the SCOBOLSOURCE attribute is:

```
SCOBOLSOURCE <file-name> [!]

  NOTE:  APPL must be the object type in effect when you
  supply a value for this application attribute.

<file-name>

  identifies the file to which ENABLE is to write the
  SCREEN COBOL source code.  ENABLE writes this file in
  edit-type file format.

!

  indicates that ENABLE is to overwrite an existing file.
  If an edit-type file named <file-name> exists and you
  specify !, ENABLE writes over the existing file when you
  generate an application with this SCOBOLSOURCE setting.
  If <file-name> already exists and ! is not specified or
  if <file-name> is not an edit-type file, an error occurs.


If you do not set a value for the SCOBOLSOURCE attribute,
ENABLE writes the source code to a temporary file with a
name of the form ZZENSnnn.  If no errors occur during
compilation, ENABLE purges this file.
```

Considerations

This command is useful if you want to modify an application generated by ENABLE. You can use the SCREEN COBOL source code produced by ENABLE, make your modifications, and compile the SCREEN COBOL source program.

To compile the SCREEN COBOL source code, you can enter the SCREEN COBOL COMPILER command from the command interpreter using the following syntax:

```
SCOBOLX /IN <source-file-name>, OUT <list-file>, MEM 64,
   NOWAIT/ <object-file-ID>

  <source-file-name>

    is the name of the file that contains the modified
    SCREEN COBOL source code.

  <list-file>

    is the name of the file that is to receive the SCREEN
    COBOL listing.

  <object-file-ID>

    is the destination of the SCREEN COBOL object code,
    of the form:

    [\<system>.][$<volume>.][<subvolume>.]<object-ID>

      \<system>     is the system on which the object files
                    reside.

      $<volume>     is the volume on which the object files
                    reside.

      <subvolume>   is the subvolume on which the object
                    reside.

      <object-ID>   is an identifier of up to five
                    characters used as the leftmost portion
                    of the object file names.  The SCREEN
                    COBOL compiler appends the suffix COD
                    or DIR to this identifier to complete
                    each file name.
```

Example

The following command identifies "$mkt.sample.scsource" as the name of the file to which ENABLE is to write the SCREEN COBOL source code:

```
SET APPL SCOBOLSOURCE $mkt.sample.scsource
```

SCREENFORMAT Attribute

The SCREENFORMAT attribute indicates the screen layout scheme for a box. The syntax for the SCREENFORMAT attribute is:

```
SCREENFORMAT { UNCOMPRESSED }
             { COMPRESSED   }
```

UNCOMPRESSED

indicates that each screen line in the box is to display at most one screen field. If the value of the HEADINGS attribute is not NULL, each screen line displays exactly one screen label.

COMPRESSED

indicates that each screen line in the box is to contain as many screen label and field pairs as will fit on a screen line. If HEADINGS is not NULL, labels and fields for the elementary items of groups begin on a new screen line. If HEADINGS is NULL no more than one record appears on each screen line. (See the SIZE attribute for more information.)

The starting value of the SCREENFORMAT attribute is UNCOMPRESSED.

Consideration

When HEADINGS is not NULL, SCREENFORMAT is COMPRESSED, and the record description associated with a box contains group fields, the application displays labels and fields on a new screen line whenever it encounters a group or a new level number within the group in the record description.

For example, consider the following record description:

```
RECORD employee.
FILE IS employee KEY-SEQUENCED.
02 empnum        PIC 9(4).
02 empname       PIC X(18).
02 dept.
   04 regnum     PIC 99.
   04 branchnum  PIC 99.
02 job           PIC X(12).
02 age           PIC 99.
02 salary        PIC 9999V99.
02 vacation      PIC 99.
KEY 0 IS empnum.
KEY "en" IS empname.
KEY "dp" IS dept.
END
```

If you generate a single-file application by using the following ENABLE commands:

```
SET BOX RECORD employee
SET BOX SCREENFORMAT COMPRESSED
ADD BOX employ-box
SET APPL TREE (01 employ-box)
SET APPL PATHCOMFILE emppath
ADD APPL employee-compres
GENERATE employee-compres
```

the application displays a screen with a compressed format. Figure 4-2 shows this screen.

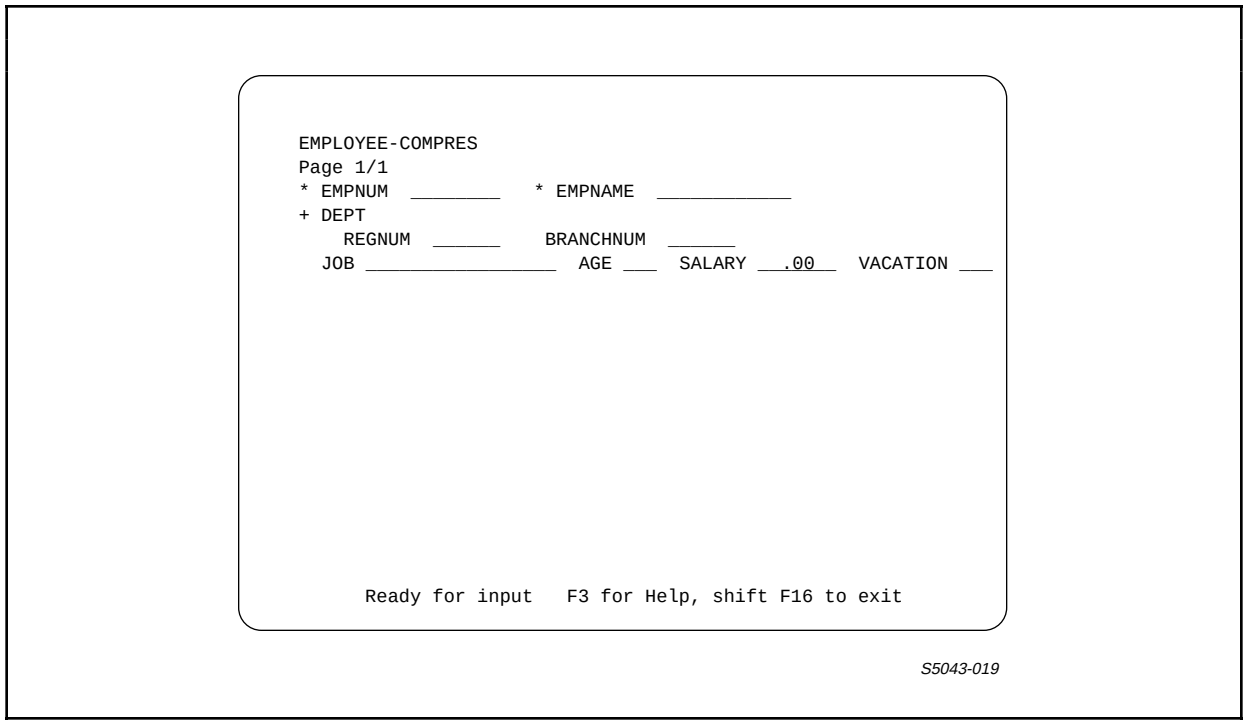


Figure 4-2. Sample Screen With SCREENFORMAT COMPRESSED

Notice that the screen labels and fields for "regnum" and "branchnum" appear on the screen line following the label for the "dept" group.

If you supply COMPRESSED as a value for SCREENFORMAT and NULL as a value for HEADINGS, the application:

- Displays as many screen fields from a record as will fit on a screen line
- Omits any group fields in the record; the application displays only the elementary items within each group
- Omits screen labels for the fields; in this case, you can provide your own column labels by using the BOXTITLE attribute

Example

The following command sets SCREENFORMAT to COMPRESSED:

```
SET BOX SCREENFORMAT COMPRESSED
```

SERVERCLASS Attribute

The SERVERCLASS attribute identifies the name of the server class to which the General Server belongs. The syntax of the SERVERCLASS attribute is:

```
SERVERCLASS <server-class-name>

<server-class-name>

  identifies the name of a server class.  ENABLE uses this
  name in the generated SCREEN COBOL program and in the
  PATHCOM command file.  When you supply a value for the
  SERVERCLASS attribute, the General Server belongs to this
  server class within the PATHWAY system.

  <server-class-name> must conform to the following naming
  conventions:

  1.  The name can be up to 15 characters in length.

  2.  The name must not be a PATHCOM reserved word.

  3.  The name must not begin with the ENABLE reserved
      prefix T9155-.

The starting value of the SERVERCLASS attribute is
ENABLE-SERVER.
```

Considerations

An application must use different copies of the General Server if any of the following are true:

- For some boxes used by an application the value of TMF is ON (indicating that the boxes represent files audited by TMF, the Transaction Monitoring Facility), and for other boxes used by the application the value of TMF is OFF.

- For some boxes used by the application the value of TMF is ON, and for other boxes used by the application the value of NONSTOP is ON (indicating that the General Server is to run as a NonStop process pair).

- For some boxes used by the application the value of NONSTOP is ON, and for other boxes the value of NONSTOP is OFF.

If these boxes are used by the same application, you must use different copies of the General Server by supplying a value for the SERVERCLASS attribute for each box.

Example

If you generate an application by including the following commands:

```
SET BOX RECORD parts
SET BOX NONSTOP ON
SET BOX SERVERCLASS my-server
ADD BOX parts-1
RESET BOX NONSTOP
SET BOX RECORD supplier
SET BOX TMF ON
SET BOX SERVERCLASS tmf-server
ADD BOX suppliers
```

a copy of the General Server belonging to a server class named "my-server" accesses the data base file represented by the "parts-1" box. A copy of the General Server belonging to a server class named "tmf-server" accesses the data base file represented by the "suppliers" box.

SIZE Attribute

The SIZE attribute specifies the number of records that the application displays within a box on the screen. The syntax of the SIZE attribute is:

```
SIZE <number>

<number>

  is a positive integer that defines the number of records
  that the application can display within the box. If you
  enter an invalid value for <number> (such as -2), ENABLE
  responds with an error message.

The starting value of the SIZE attribute is 1.
```

Considerations

This command is useful if you want to display more than one record occurrence in a box. You can generate an application that displays a tabular screen by setting SIZE to an appropriate value and also setting SCREENFORMAT to COMPRESSED, HEADINGS to NULL, and use BOXTITLE(s) to display appropriate screen labels.

Maximum Size. ENABLE cannot generate an application if the number of data characters within a box is greater than 2,048. To determine the number of data characters in a box, multiply the number of characters per record by the value for SIZE.

DDL OCCURS Clause Nesting. When the SIZE attribute is set to a value greater than one, OCCURS clause items can be nested to only three levels. When set to one, OCCURS items can be nested to four levels. Refer to the Data Definition Language (DDL) Reference Manual for more information.

Multifile Considerations. If you want to display more than one record in a box for a multifile application, consider the relationship between the records in the parent and child boxes when you supply a value for SIZE:

- If a one-to-one relationship exists between the parent and child box, a value of one should be supplied for both parent and child. A one-to-one relationship exists if one record for a child box has the same join field value as one record for a parent box.

- If a one-to-many relationship exists between the parent and child box, SIZE should be 1 for the parent box and can be greater than one for the child box. A one-to-many relationship exists if several records for a child box can can have the same join field value as a single record for a parent box.

- If a many-to-one relationship exists between a parent and a child box, SIZE should be 1 for the child box and can have a value greater than 1 for the parent box. If SIZE is greater than 1 for the parent box, you should supply OFF as the value of the DELETE, INSERT, and UPDATE attributes for the child box. A many-to-one relationship exists when several records from a parent box have the same join field value as a single record from a child box.

Example

If you execute an application generated with the following series of ENABLE commands:

```
SET BOX RECORD parts
ADD BOX part1
SET BOX RECORD supplier
ADD BOX supplier-1
SET BOX SIZE 2
SET BOX RECORD orders
ADD BOX orders
...
```

the application displays the screen label and field pairs for one record in the "Part1" and "supplier-1" boxes and displays the screen labels and field pairs for two records in the "supplier-1" box.

TERMINAL Attribute

The TERMINAL attribute identifies the type of terminal upon which the application can run. The syntax of the TERMINAL attribute is:

```
TERMINAL <terminal-type>

  NOTE:  APPL must be the object type in effect when you
  supply a value for this application attribute.

<terminal-type>

  identifies the type of terminal upon which the
  application can run.  Valid values for <terminal-type>
  are:

  • T16-6510
  • T16-6520
  • T16-6530
  • IBM-3270

The starting value of the TERMINAL attribute is the type of
terminal from which you enter the ENABLE commands.
```

Considerations

If you supply T16-6520 as a value for the TERMINAL attribute, the application will run on a Tandem 6530 terminal as well as on a Tandem 6520 terminal.

If you supply T16-6530 as a value for the TERMINAL attribute, the application will run on a Tandem 6520 terminal as well as on a Tandem 6530 terminal. When you supply T16-6530 as a value for the TERMINAL attribute, the generated SCREEN COBOL source code states that the terminal type is T16-6520.

Example

The following command sets the TERMINAL attribute to T16-653x:

```
SET APPL TERMINAL T16-6530
```

TITLE Attribute

The TITLE attribute specifies the title that appears left-justified on the first line of each screen. The syntax of the TITLE attribute is:

TITLE <string-literal>
NOTE: APPL must be the object type in effect when you supply a value for this application attribute.
<string-literal>
is the screen title. A <string-literal> must be enclosed in quotation marks (" ") and can be up to 79 characters long. If you specify a <string-literal> that is longer than 79 characters, ENABLE issues an error message.
Embedded quotation marks (") are not allowed.
The starting value of the TITLE attribute is null. ENABLE uses the application name as the default value of this attribute when you add the application.

Consideration

If an application displays more than one page, the screen title appears on the first line of each page.

Example

The following command sets the value of the TITLE attribute to ENTRY SCREEN:

```
SET APPL TITLE "ENTRY SCREEN"
```

TMF Attribute

The TMF attribute indicates whether the data base file represented by the box is, or is not, audited by the Transaction Monitoring Facility (TMF). The syntax of the TMF attribute is:

```
TMF { ON  }
    { OFF }
```

ON

indicates that the data base file is audited by TMF.

OFF

indicates that the data base file is not audited by TMF.

The starting value of the TMF attribute is OFF.

Considerations

When you supply ON as a value for the TMF attribute:

1. ENABLE generates the application so that BEGIN-TRANSACTION and END-TRANSACTION statements appear in the SCREEN COBOL code that controls access to the data base file.

2. The generated application can only be used on a system for which TMF is active.

3. The copy of the General Server that accesses the file represented by the box must be defined as TMF. This means that the PATHCOM command file must contain the following command:

   ```
   SERVER (PARAM TMF ON)
   ```

4. The generated PATHCOM command file indicates that the TCP is to run as a NonStop process pair.

For the same box, you cannot supply ON as the value for both TMF and NONSTOP. ENABLE allows you to generate an application that uses some boxes with TMF ON and other boxes with TMF OFF. If you do this, the value of the SERVERCLASS attribute must be different for both categories of boxes.

Example

Given an application generated with the following commands:

```
SET BOX TMF ON  <------------  supplies ON as the value for
SET BOX SERVERCLASS tmf-serv   TMF
SET BOX RECORD parts
ADD BOX parts-1
SET BOX RECORD supplier
ADD BOX suppliers
SET BOX TMF OFF  <-----------  supplies OFF as the value for
SET BOX SERVERCLASS ord-serv   TMF
SET BOX RECORD order
ADD BOX orders
```

the boxes "parts-1" and "suppliers" both have TMF ON, and both have the same value for the SERVERCLASS attribute: "tmf-serv." The "orders" box, which has TMF OFF, has a different value for the SERVERCLASS attribute: "ord-serv."

TREE Attribute

The TREE attribute associates a box, or set of boxes, with an application. It specifies a hierarchical relationship between the boxes. Based on this logical structure, the application determines the access path to records for display within a child box. The syntax of the TREE attribute is:

```
TREE ( <level> <box-name>

  [ <level> <box-name> <link-optional-clause> ] ... )

  NOTE:  APPL must be the object type in effect when you
  supply a value for this application attribute.

<level>

  is a positive integer from 1 to 50 that indicates the
  hierarchical position of the <box-name> it preceeds.

  The first box must have the lowest <level>.  Subsequent
  boxes must have higher level numbers.  These level numbers
  need not be consecutive.

  A child box must have a higher <level> than the parent
  associated with it.  All child boxes associated with the
  same parent must have the same <level>.

<box-name>

  is the name of a box that must appear in the object table
  when the application is added.  For single-file
  applications, this name may be supplied at the time the
  application is generated.
```

--->**>**

<link-optional-clause>

is one of:

LINK <parent-join-field> TO OPTIONAL <child-join-field>

or

LINK <box-name> TO OPTIONAL <box-name> VIA <join-field>

The <link-optional-clause> associates a parent box with a child box, and is further described under Considerations.

<parent-join-field>

is the name of a field through which the parent box is linked to a child box.

<child-join-field>

is the name of a field through which a child box is linked to its immediate parent-box. The <child-join-field> should be a key field, the leading (leftmost) portion of a composite key group, or a courtesy key (the record number of an entry-sequenced, relative, or unstructured file). If the <child-join-field> is a courtesy key, it must be identified by the keyword KEY. A primary key can also be identified by the keyword KEY.

<join-field>

should be a key field, the leading (leftmost) portion of a composite key field, or a courtesy key (the record number of an entry-sequenced, relative, or unstructured file) for the child box. If <join-field> is a courtesy key, it must be identitied by the keyword KEY.

The function of join fields is described in detail under Considerations.

--->

4-72

The starting value of the TREE attribute is null. For single-file applications, ENABLE uses "(01 <application-name>-BOX)" as the default value of the TREE attribute, where <application-name> is the name specified when you enter either an ADD APPL or a GENERATE command.

Considerations

Although you can supply a value for the TREE attribute regardless of the contents of the object table, the boxes identified in the structure must exist in the object table when the application is added or generated. Refer to the ADD command for information about adding boxes to the object table.

Single-File Applications. ENABLE can supply a default value for the TREE attribute if you are generating a single-file application. ENABLE uses the application name to determine a box name. ENABLE checks to see if a box with this name exists in the object table. If the named box is found, ENABLE uses the box for the application. If the box is not found, ENABLE attempts to add a box based on the current contents of the attribute table.

Multifile Applications. Multifile applications incorporate several associated boxes according to the hierarchical structure specified with the TREE attribute. ENABLE uses this structure to determine:

- The links that connect boxes to each other (Each box in the structure must be linked to at least one other box.)
- The order of appearance and nesting of boxes on the terminal screen (One or more fields from the first box must be included in the display.)
- The order in which the application is to retrieve or insert records for the boxes (An application must insert or read a record for a parent box before it can insert or read a record for a child box.)

The tree structures for several multifile applications are symbolically represented in Figure 4-3.

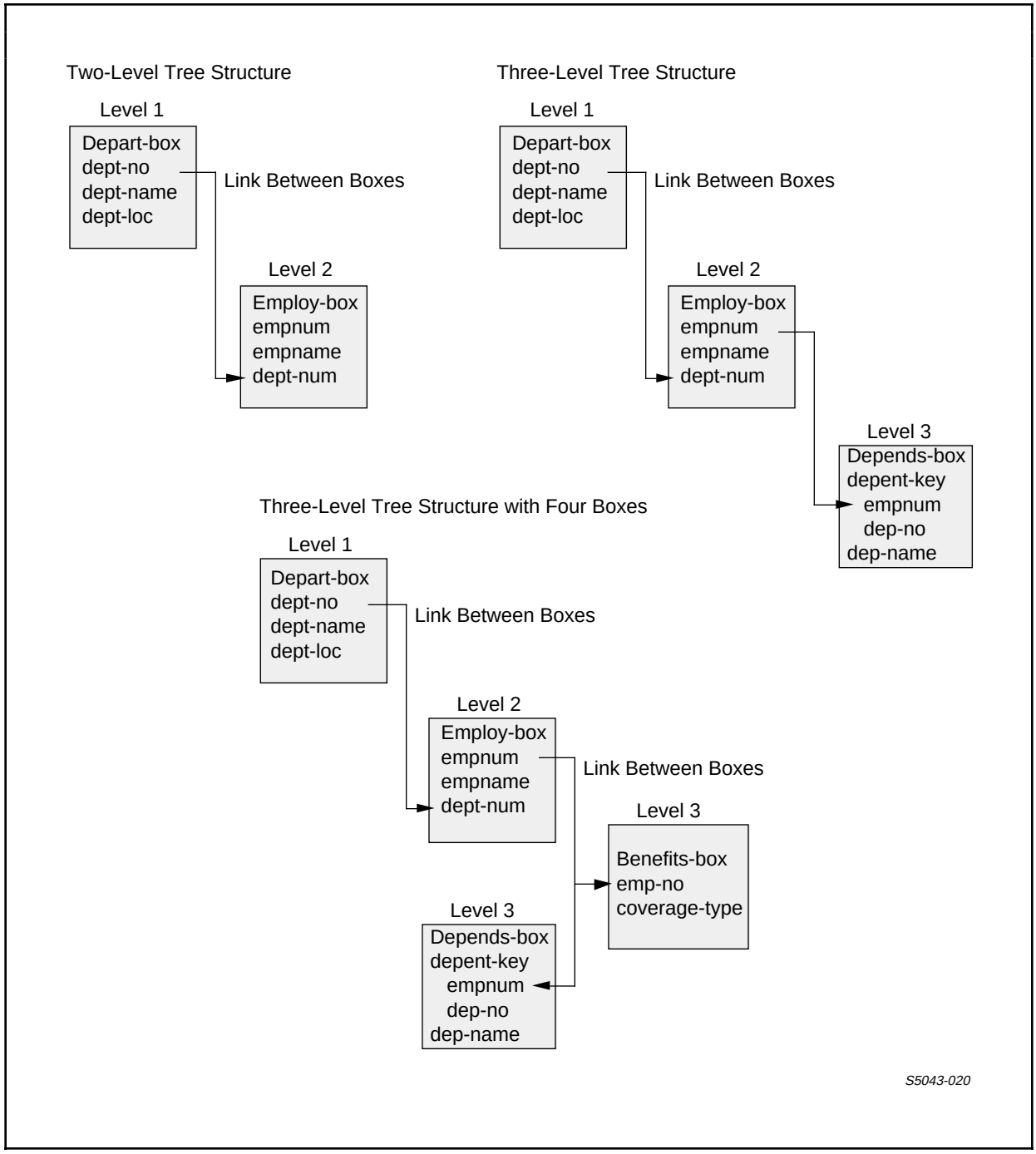


Figure 4-3. Symbolic Representation of Tree Structures

What is a LINK? A link is the portion of a tree structure that defines the logical connection between two boxes. While a tree structure can contain multiple links that connect the same box to several other boxes, a single link connects only two boxes.

To establish a link between two boxes in the tree structure, you identify matching fields, called join fields, for both boxes. When ENABLE generates an application, it includes program logic that tells the application to use the data values in these join fields to insert and read records for both boxes.

Figure 4-4 shows the records associated with two boxes: "employee-box" and "depend-box." This figure also shows some matching join field values in these records. In this figure, the join fields are "empno" and "empnum."

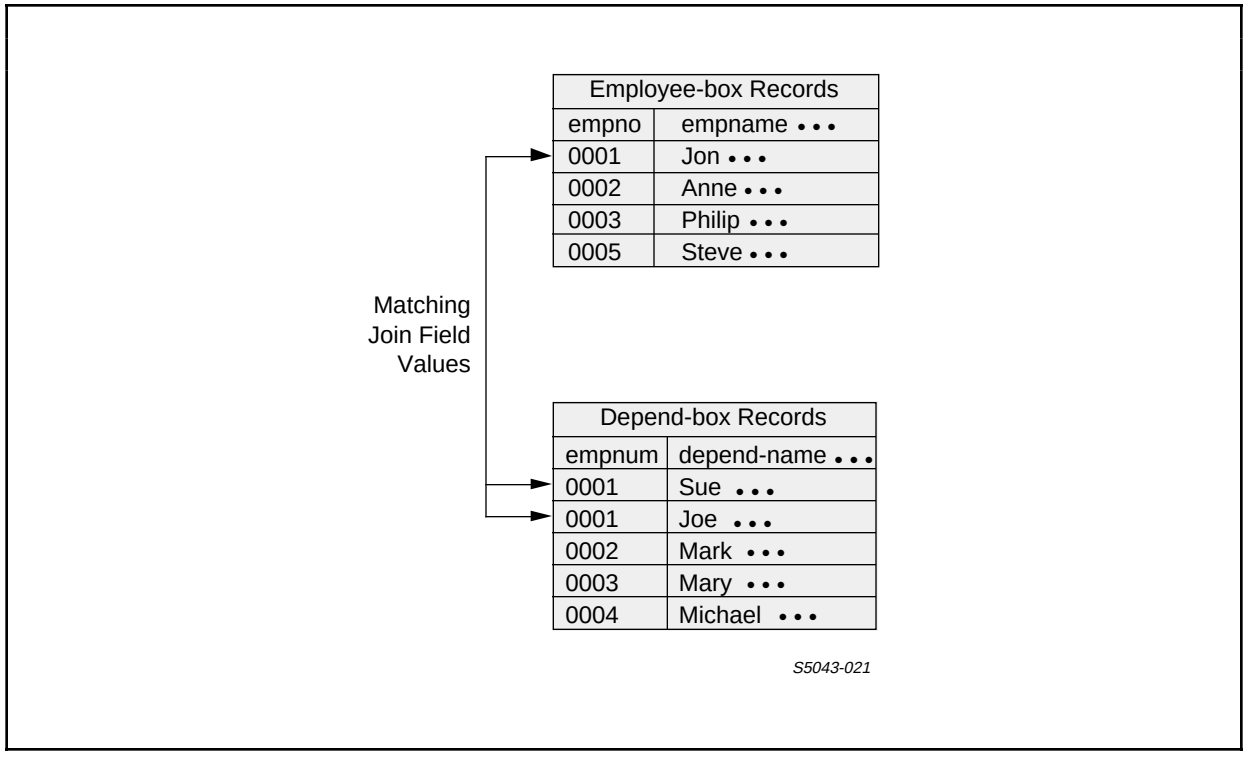


Figure 4-4. Example of Matching Join Fields in Records

Because a tree is a hierarchical structure, the link provided by ENABLE is a special link called a LINK OPTIONAL. When you establish a LINK OPTIONAL for two boxes, you define the order in which the application is to read or insert records for the boxes as well as identifying the join fields for the boxes. For this reason, a LINK OPTIONAL is an ordered link.

The order that you define with a LINK OPTIONAL establishes a dependency between the records associated with both boxes. When you define a LINK OPTIONAL, you indicate that the application is to use the join field value of a record from one box to read or insert a record in the other box. Thus, the records in the second box can be said to depend on a matching record in the first box for access by the application. Records match if their join field values are the same.

Since the dependency established for a LINK OPTIONAL is similar to the dependency condition that exists between a parent and a child, one of the boxes connected by a LINK OPTIONAL is called a parent box and the other is called a child box. In this context, the application can read or insert a record for a child box only if it has already read or inserted a matching record for the parent box.

Figure 4-5 illustrates an application that uses "employee-box" as the parent box and "depend-box" as the child box. (Figure 4-4 shows part of the records associated with these boxes.)

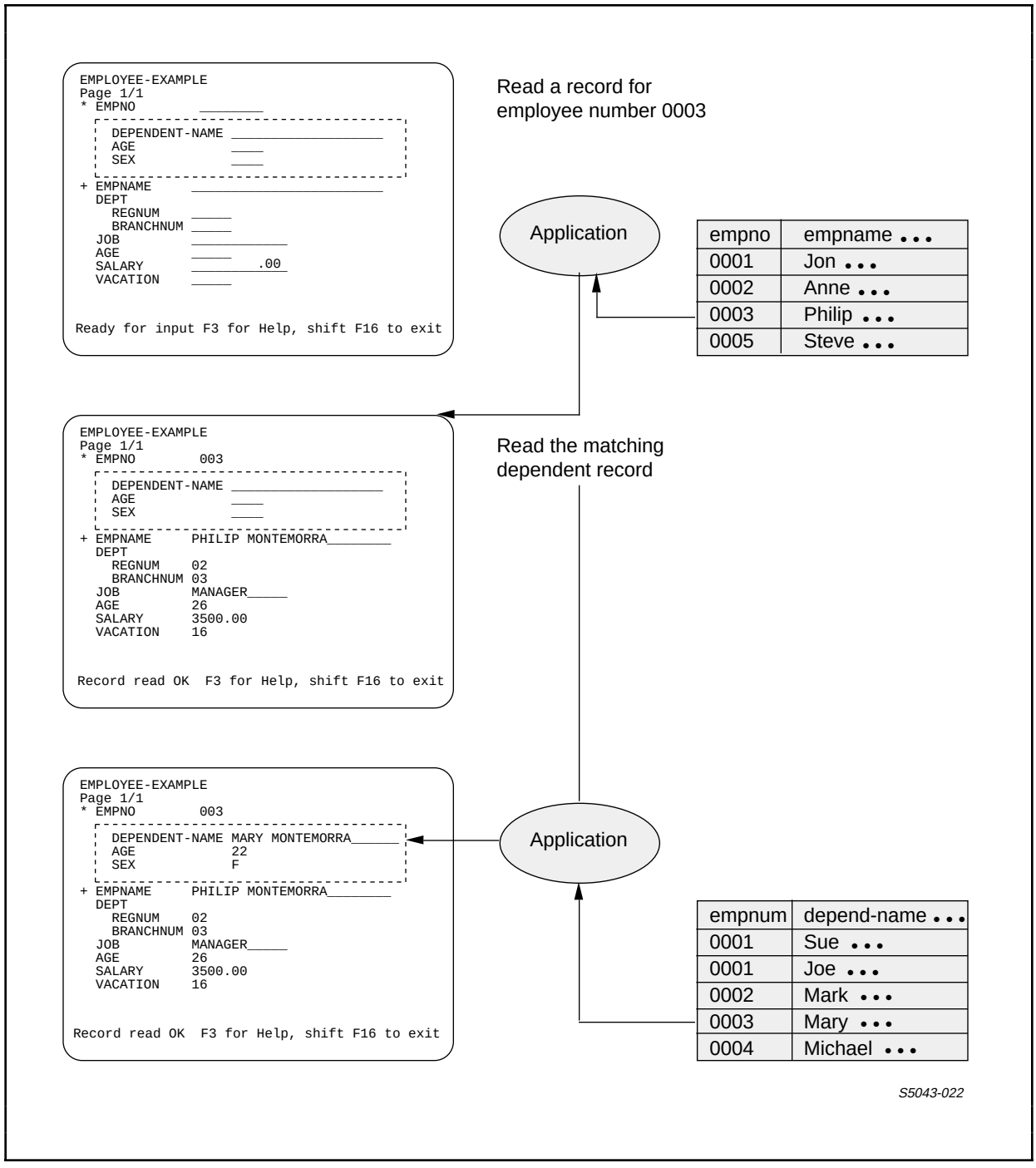


Figure 4-5. Reading Records for Parent and Child Boxes

The screen displayed by an application reflects the parent-child relationship established by a LINK OPTIONAL. On the screen, an application always displays the join field of the parent box on the screen line that directly precedes the child box. The application does not display the join field of the child box. Figure 4-6 shows the screen displayed by an application that uses "employee-box" as the parent and "depend-box" as the child.

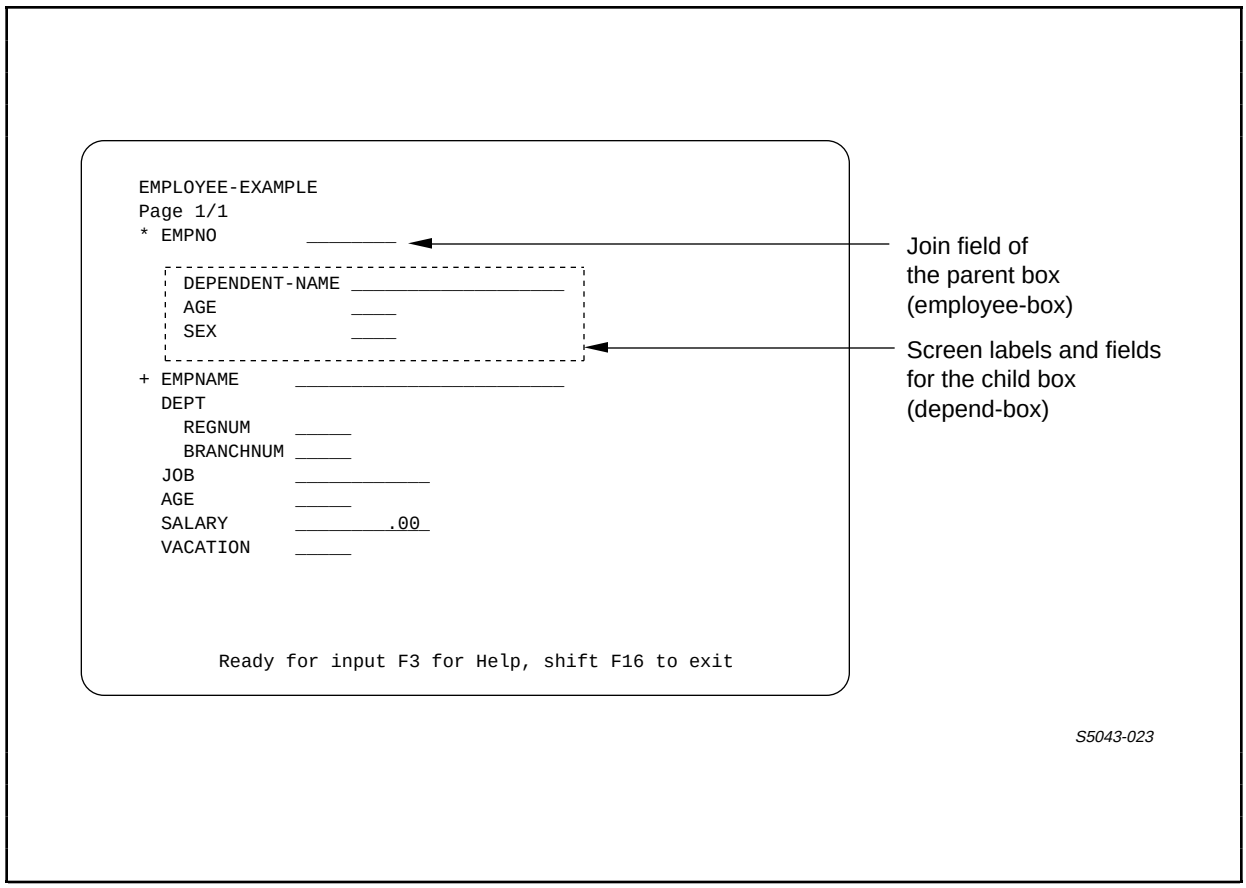


Figure 4-6. Sample Terminal Screen With Parent and Child Boxes

Before you can establish a link between two boxes, you must first:

- Determine if each box has an appropriate join field
- Determine which box is to be the parent box and which is to be the child box

Determining Appropriate Join Fields. To find potential join fields, examine the record descriptions associated with both boxes. These record descriptions must describe matching fields. Fields match if they have compatible data types.

You obtain the data type of a field from the PIC clause associated with that field in the record description. If the data types of both fields are compatible, the fields are potential join fields. Table 4-3 lists the compatible data types.

Table 4-3. Compatible Data Types

Data Type	Compatible Data Types
Alphabetic (PIC A)	Alphabetic (PIC A) Alphanumeric (PIC X or TYPE CHARACTER)
Alphanumeric (PIC X or TYPE CHARACTER)	Alphabetic (PIC A) Alphanumeric (PIC X or TYPE CHARACTER) Numeric Integer (PIC 9, PIC 9 COMP or TYPE BINARY) Numeric Noninteger (PIC 9 V 9.. or Scaled item)
Numeric Integer (PIC 9, PIC 9 COMP or TYPE BINARY)	Numeric Integer (PIC 9, PIC 9 COMP or TYPE BINARY) Numeric Noninteger (PIC 9 V 9.. or Scaled item) Alphanumeric (PIC X or TYPE CHARACTER)
Numeric Noninteger (PIC 9 V 9.. or Scaled item)	Numeric Integer (PIC 9, PIC 9 COMP or TYPE BINARY) Numeric Noninteger (PIC 9 V 9.. or Scaled item) Alphanumeric (PIC X or TYPE CHARACTER)
Note that the data type of a courtesy key is Numeric Integer.	

To determine if potential join fields are appropriate, consider the data values that the fields represent. Appropriate join fields should represent the same data values. Although ENABLE will allow you to link two boxes whose join fields represent different kinds of data values, avoid doing this because an application generated with such a link might not operate in the manner you expect.

Figure 4-7 shows the record descriptions associated with the boxes "employee-box" and "depends-box" discussed earlier in this section.

Record Description Associated With Employee-box	Record Description Associated With Depends-box
RECORD employee.	RECORD dependents.
FILE IS employ KEY-SEQUENCED.	FILE IS depend RELATIVE.
02 empno PIC 9(4).	02 empnum PIC 9(4).
02 empname PIC X(30).	02 dependent-name PIC X(30).
02 dept.	02 age PIC 9(2).
04 regnum PIC 9(2).	02 sex PIC X.
04 branchnum PIC 9(2).	KEY "ep" IS empnum.
...	KEY "dn" IS dependent-name.
KEY 0 IS empno.	KEY "ag" IS age.
KEY "en" IS empname.	END
END	

Figure 4-7. Sample Record Descriptions

Notice that the "empno" field of "employee-box" has a data type that is compatible with the "empnum" field of "depends-box." The data types of these fields are compatible because both are numeric integer. The names of these fields indicate that each represents an employee identification number. If the files described by the boxes contain data, you should examine this data to determine if potential join fields represent the same data values. Because both "empno" and "empnum" have compatible data types and represent the same data values, these fields are appropriate join fields for "employee-box" and "depends-box."

Choosing the Parent and Child Box. To determine which box is to be the parent box and which box is to be the child box, consider the purpose for which you are generating the application. Under most circumstances, you should make the box associated with the records in which you are most interested the parent box. Remember, the application cannot read or insert a record for the child box unless a matching record exists for the parent box.

When you decide which box is to be the parent box and which is to be the child, you also affect the appearance of the terminal screen displayed by an application. Remember, an application always displays the join field of the parent box on the screen line that directly precedes the child box.

ENABLE has two other requirements that should affect your decision. These requirements are:

1. The size of the join field of a parent box must be smaller than or equal to the size of the join field for a child box.

   To determine the size of a join field, examine the PIC clause of the field in the record description. Besides identifying the data type of the field, this clause also defines the number of characters that the field can store. Consider, for example, the following PIC clauses and the corresponding number of characters that they define for a field:

PIC Clause	Number of Characters in the Field
PIC 9(4).	4 characters
PIC X(20).	20 characters
PIC 9(4)V99.	6 characters
PIC A(30).	30 characters

   For COMP fields the number of 9s represented by the PIC clause determines the number of characters in the field as follows:

Number of 9s	Number of Characters
1-4 (For example, PIC 9(3))	2
5-9 (For example, PIC 9(6))	4
10-18 (For example, PIC 9(11))	8

2. The join field of a child box must be a primary key field, an alternate key field, a courtesy key field, or the leading (leftmost) portion of a composite key field.

   Alternate keys are not recognized within child boxes. To provide a key for the subset of records that match the join field within the child box, use the leftmost portion of a composite key field for the parent join field. Use the entire composite key field for the child join field. The remaining (rightmost) portion of the composite key is then recognized as a key for the matching records in the child box.

After you identify links to connect the boxes within the tree structure, you can determine the level at which each box is to reside in the tree structure.

Levels of a Tree Structure. Levels are the portions of the tree structure that determine the order in which boxes are used by an application. ENABLE uses the levels to determine:

- The order in which boxes appear on the screen displayed by the application. For example, the application displays a field or fields from the box at the highest level of the tree before it displays a field from any other boxes used by the application.

- The order in which the application can read and insert records for the boxes. For example, the application must read or insert a record from the box at the highest level of the tree before it can read or insert records for any other boxes.

The simplest tree structure, built for a single-file application, has only one level. More complex tree structures built for multifile applications can have several levels. Each box used by an application must reside at a specific level within the tree structure.

A box that resides at the first (lowest) level of the tree structure is the parent of all other boxes used by an application. Only one box can reside at this level.

You must provide a link that connects the box at the first level of the tree to one or more boxes at the second level. You can link the boxes at the second level to other boxes at subsequent levels of the tree structure. You cannot, however, link boxes at the same level of the tree structure to each other.

Since you connect the levels of the tree structure with links, you must define the level of a parent box as being lower than its corresponding child box. Depending on how you define the links and levels within your tree structure, a box can be the parent of several boxes at a higher level of the tree structure and the child of a box in a lower level of the tree. Although a box can be the parent of several other boxes, a box cannot be the child of more than one box.

When you define the level at which a box resides in the tree structure, ENABLE uses this information to generate the program logic for the application. This logic indicates:

- For the box at the lowest level, the application can read or insert any valid record.
- For boxes at all other levels, the application must read or insert a record for the parent box (which must reside at a lower level of the tree) before it can read or insert a matching record for the child box.

In summary, the following rules apply to the levels of a tree structure:

- Only one box can reside at the lowest or first level of the tree.
- Boxes that are connected by a link cannot reside at the same level of the tree structure.
- The level at which a parent box resides must be lower than the level at which a child box resides.
- A box can be the parent of several boxes at higher levels of the tree structure; however, a box cannot be the child of more than one box at a lower level of the tree structure.

When you define the levels at which boxes reside in the tree structure, these levels affect the appearance of the terminal screen displayed by the application and also determine the records that can be read or inserted for each box used by the application.

Using the TREE Attribute. The following rules apply when you use the TREE attribute:

- The name of the box at the first level of the tree structure must appear before any other box names in the tree structure.

- The level for the box at the first level of the tree structure must be lower in value than any other level that appears in the tree structure.

- When you define a level for the box at the first level of the tree, do not include the LINK OPTIONAL.

- When you define level numbers for boxes at subsequent levels of the tree structure, these level numbers must be higher in value than any box defined at a lower level in the tree structure.

- You must include LINK OPTIONAL when you define the level of a box at any but the first level of the tree structure.

- When you use LINK OPTIONAL, you must identify the parent box before you identify the child box. To do this, you can use either of the following forms of LINK OPTIONAL:

  LINK parent-join-field TO OPTIONAL child-join-field

  LINK parent-box TO OPTIONAL child-box VIA join-field

  Use the second form of this option only when the join fields of both boxes have the same field name.

Examples

The following examples describe values of the TREE attribute that build several different tree structures.

Example of the TREE Attribute With Two Boxes. Building a tree structure for two boxes is a fairly simple procedure. You must, of course, identify appropriate join fields for each box and decide which box is to be the parent box and which box is to be the child. The box that you choose for the parent should be at the first level of the tree structure.

Suppose, for example, that you want to generate an application that displays information about departments and the employees that belong to those departments. This application must use two boxes:

- "depart-box," which represents the file that stores department information
- "employ-box," which represents the file that stores employee information

Figure 4-8 shows a portion of the record descriptions associated with these boxes and the partial contents of the files that the boxes represent.

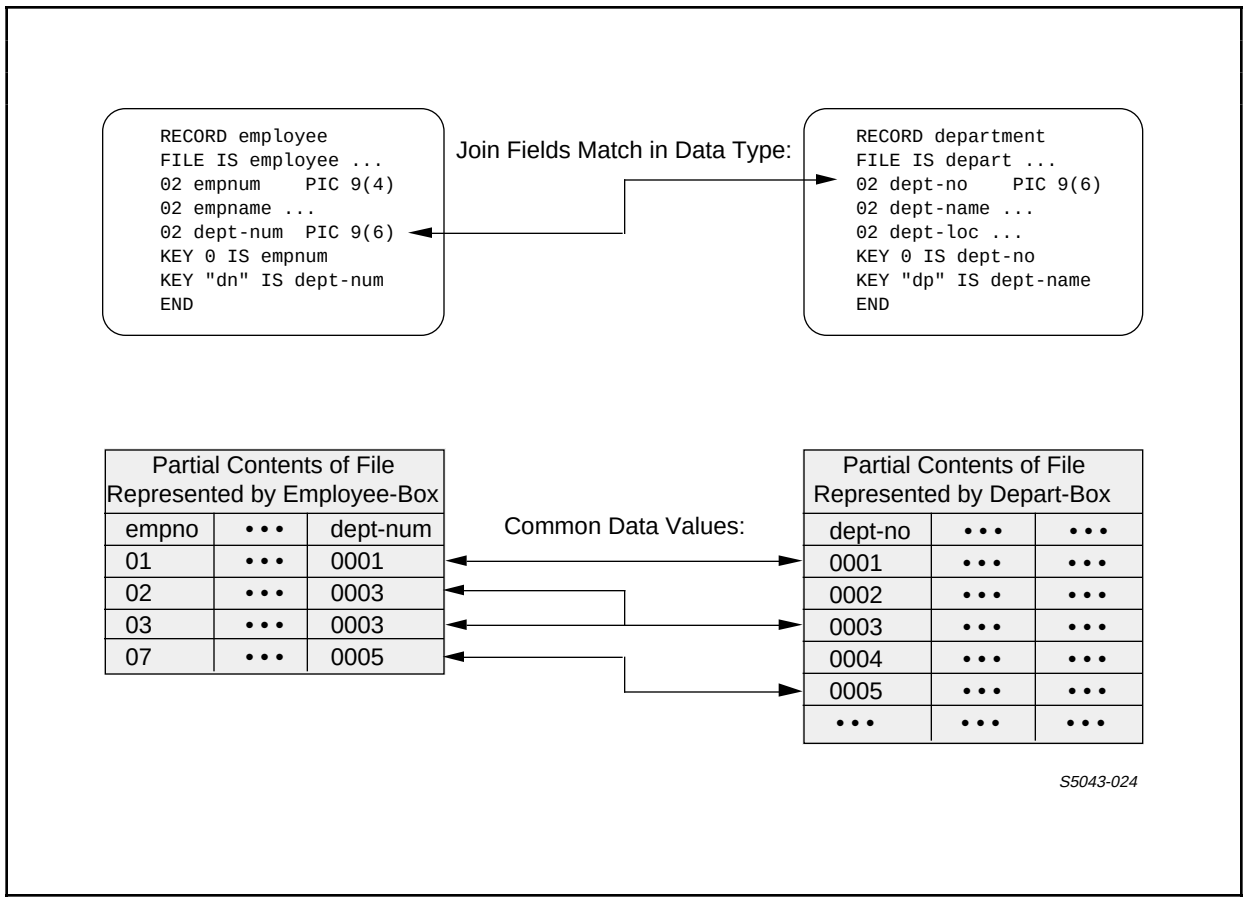


Figure 4-8. Sample Record Descriptions and File Contents for Two Boxes

Since the application is to display information about departments and the employees assigned to those departments, "depart-box" should be the parent box and "employ-box" the child box. The join fields of these boxes are "dept-no" and "dept-num" respectively.

To build a tree structure for this application, set a value for the TREE attribute as follows:

1. Identify "depart-box" as being at the first level of the tree structure:

```
SET APPL TREE (01 depart-box
```

   Remember that you do not include LINK OPTIONAL with the box at the first level of the tree structure.

   "01" is an arbitrary choice for the level number of this box. This level could be any value less than fifty, but must be the lowest number in the tree value.

2. Identify "employ-box" as being at the second level of the tree structure:

```
SET APPL TREE (01 depart-box
                  03 employ-box
```

   Note that 03 can be used as the level for this box because ENABLE does not require levels to be numbered consecutively.

3. Include LINK OPTIONAL with "employ-box." This option must be included because "employ-box" is not at the first level of the tree. Since the join field of "depart-box" is "dept-no" and the join field of "employ-box" is "dept-num," LINK OPTIONAL should appear as:

```
LINK dept-no TO OPTIONAL dept-num
       |                    |
       |                join field of the child box
       |
  join field of the parent box
```

   To complete the TREE attribute, you include LINK OPTIONAL with the part of the TREE attribute that defines the level of "employ-box" as follows:

```
SET APPL TREE (01 depart-box
                  02 employ-box
                     LINK dept-no TO OPTIONAL dept-num)
```

This value for the TREE attribute builds the tree structure shown in Figure 4-9.

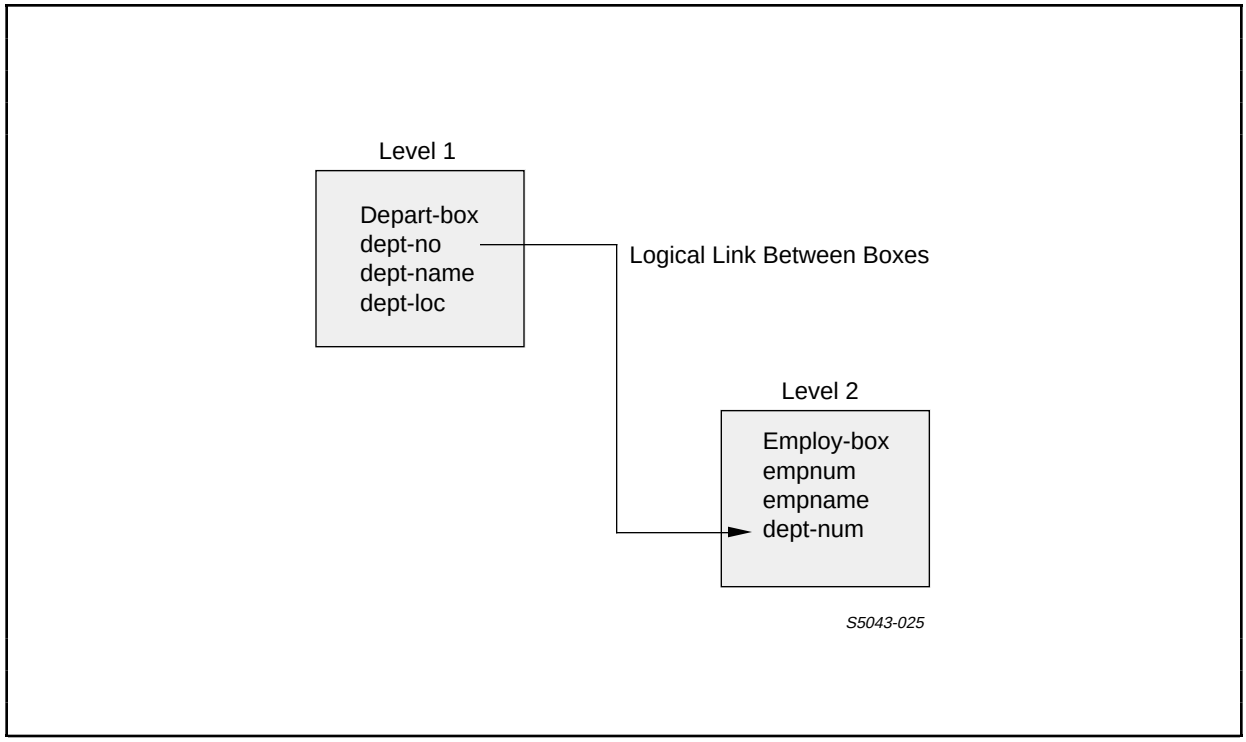


Figure 4-9. Sample Tree Structure for Two Boxes

If you use the following ENABLE commands to generate an application:

```
ADD BOX depart-box, RECORD department
ADD BOX employ-box, RECORD depart-box
SET APPL PATHCOMFILE p6
SET APPL TREE (01 depart-box
                  02 employ-box
                     LINK dept-no TO OPTIONAL dept-num)
ADD APPL dept-employ
GENERATE APPL dept-employ
```

upon execution, the application displays the screen shown in Figure 4-10.

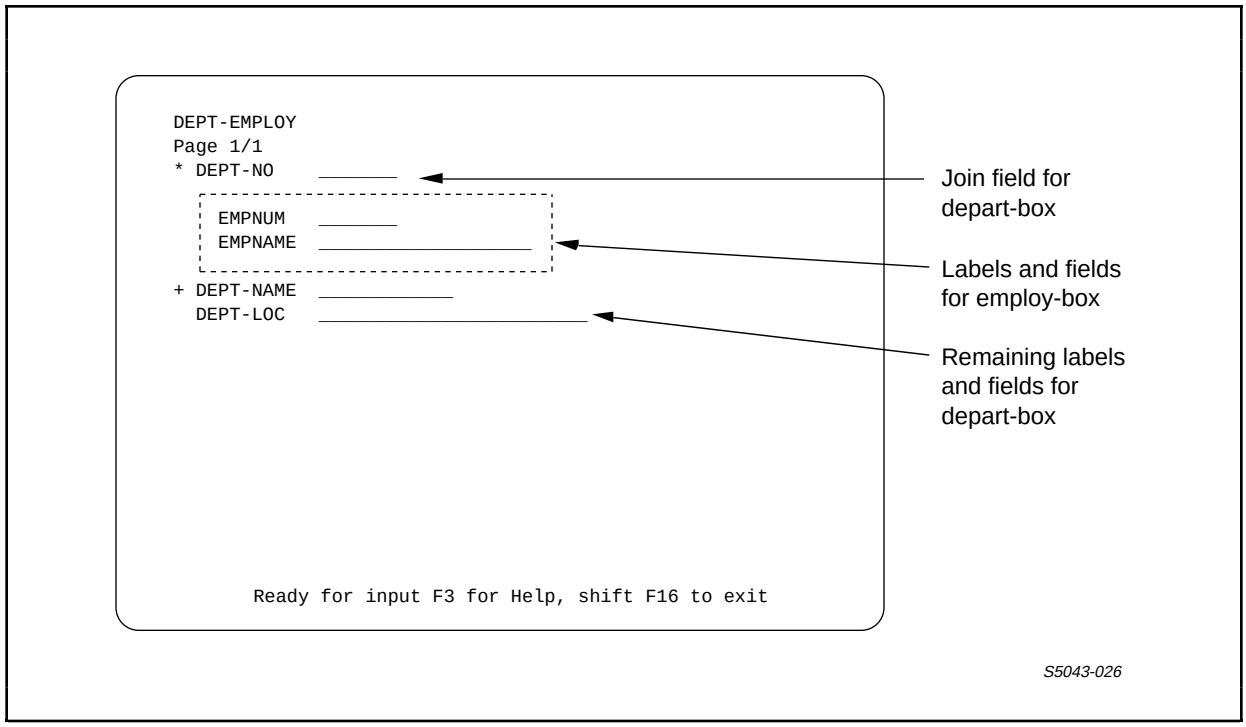


Figure 4-10. Sample Terminal Screen With Two Boxes

Notice that the join field ("dept-no") of "depart-box" appears on the screen on the screen line that directly precedes the fields for "employ-box." The join field for "employ-box" ("dept-num") does not appear on the screen since it always has the same value as dept-no.

Example of the TREE Attribute With Three Boxes. When you build a tree structure for three boxes, you should first determine the links that can exist between the boxes, then decide which box (or boxes) should be a parent box, and finally choose the level at which these boxes are to reside in the tree structure.

Suppose, for example, that you want to generate an application that displays information about departments, the employees that belong to those departments, and the dependents of those employees. Suppose further that this application is to use the following boxes:

- "depart-box," which represents a file that contains information about departments

- "employ-box," which represents a file that contains information about employees
- "depends-box," which represents a file that contains information about the dependents of employees

Figure 4-11 shows part of the record descriptions for these boxes and also shows the partial contents of the files represented by the boxes.

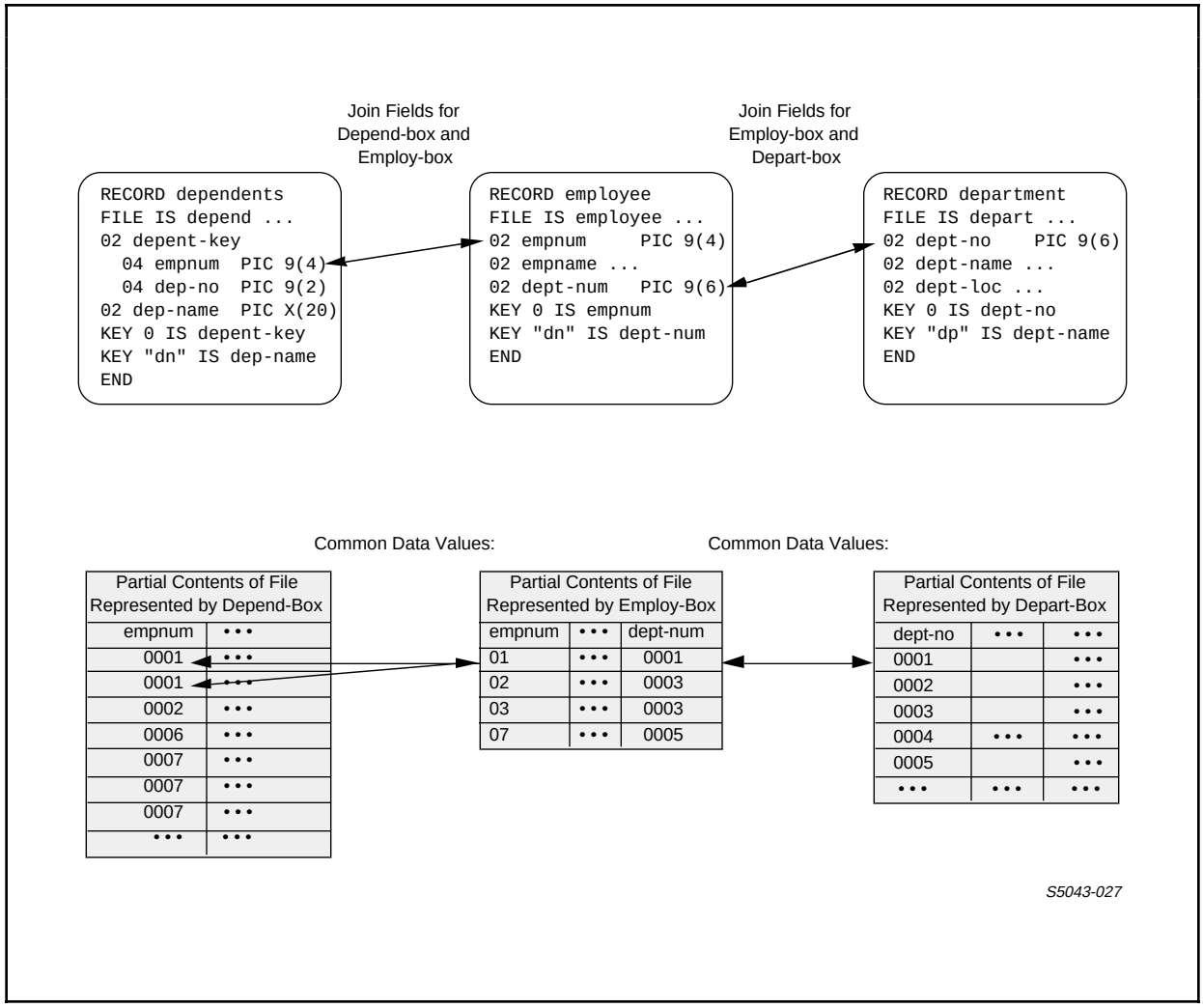


Figure 4-11. Sample Record Descriptions and File Contents for Three Boxes

Depending on the purpose of the application, any of the three boxes could be at the first level of the tree structures. These boxes could be at the first level because appropriate join fields exist to link the boxes together. Because the main purpose of the application is to display information about departments, "depart-box" should be at the first level of the tree structure. If the main purpose of the application were to display information about employees, you would make "employ-box" the box at the first level of the tree structure.

Having chosen "depart-box" as the box at the first level of the tree structure, you must now choose a box (or boxes) to reside at the second level of the tree structure. Since you cannot establish a link between "depart-box" and "depends-box" (appropriate join fields do not exist) but you can establish a link between "depart-box" and "employ-box," "employ-box" is the only box that can reside at the second level of the tree. It then follows that "depends-box" must reside at the third level of the tree structure.

To build the tree structure for this application, you set a value for the TREE attribute as follows:

1. Identify "depart-box" as being at the first level of the tree structure:

```
SET APPL TREE (02 depart-box
```

2. Identify "employ-box" as being at the second level of the tree structure:

```
SET APPL TREE (02 depart-box
                  04 employ-box
```

3. Provide the LINK OPTIONAL that connects "employ-box" to "depart-box":

```
LINK dept-no TO OPTIONAL dept-num
        |                    |
        |                join field of "employ-box"
        |
    join field of "depart-box"
```

4. Include the LINK OPTIONAL with the part of the SET APPL TREE command that identifies a level for "employ-box":

```
SET APPL TREE (02 depart-box
                  04 employ-box
                     LINK dept-no TO OPTIONAL dept-num
```

5. Identify "depends-box" as being at the third level of the tree:

```
SET APPL TREE (02 depart-box
               04 employ-box
                  LINK dept-no TO OPTIONAL dept-num
                  06 depends-box
```

6. Provide the LINK OPTIONAL that connects "depends-box" to "employ-box":

```
LINK employ-box TO OPTIONAL depends-box VIA empnum
          |                          |           |
          |                          |      Name of join
          |                          |      field for both
          |                          |           boxes
          |                          |
          |                Name of child box
          |
Name of parent box
```

7. Include the LINK OPTIONAL in Step 6 with the part of the TREE value that defines a level for "depends-box":

```
SET APPL TREE (02 depart-box
               04 employ-box
                  LINK dept-no TO OPTIONAL dept-num
                  06 depends-box
                     LINK employ-box
                     TO OPTIONAL depends-box
                     VIA empnum)
```

The preceding TREE attribute builds the tree structure shown in Figure 4-12.

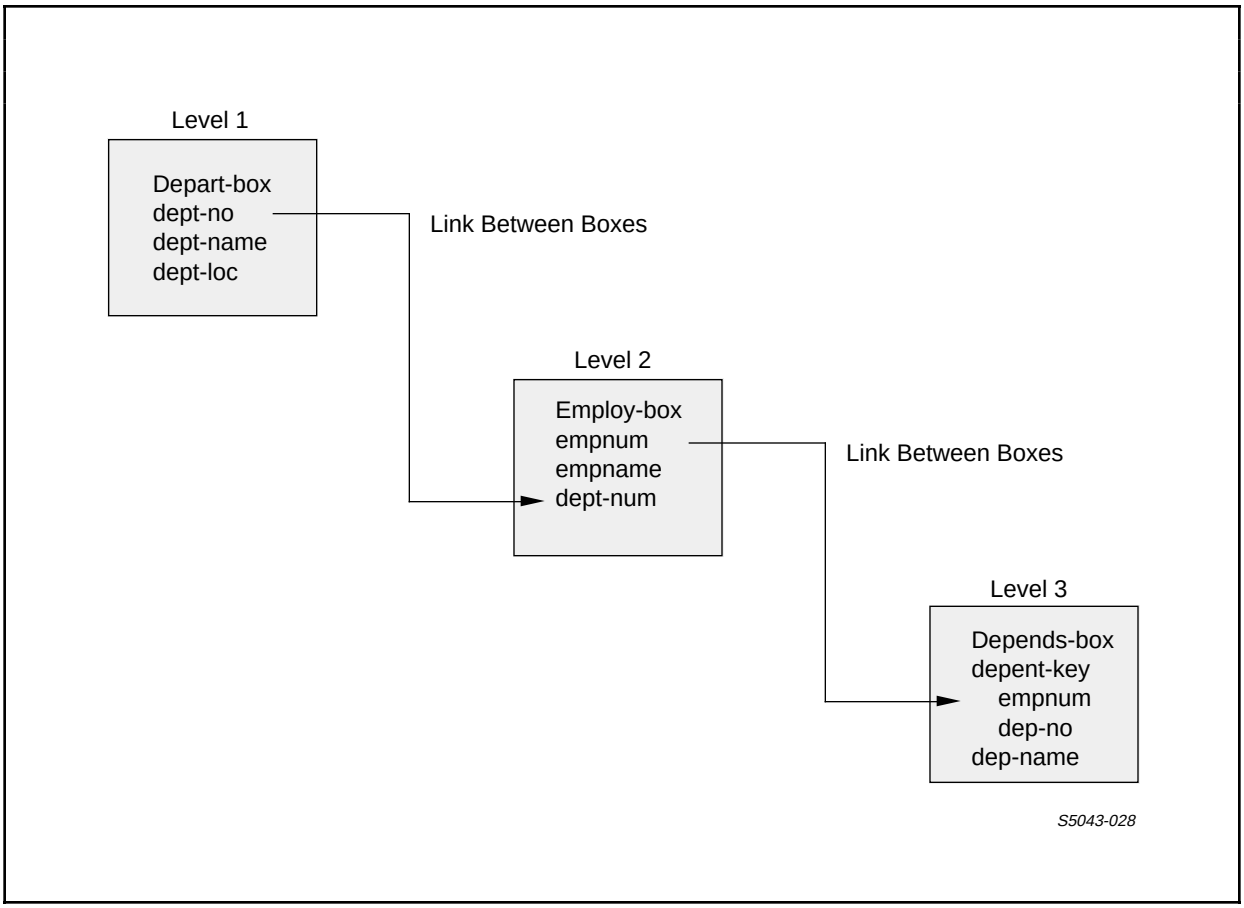


Figure 4-12. Sample Tree Structure With Three Boxes

You could generate an application with the tree structure shown in Figure 4-12 by entering the following ENABLE commands:

```
ADD BOX depart-box, RECORD department
ADD BOX employ-box, RECORD employee
ADD BOX depends-box, RECORD dependents
SET APPL PATHCOMFILE p7
SET APPL TREE (02 depart-box
                  04 employ-box
                     LINK dept-no TO OPTIONAL dept-num
                     06 depends-box
                        LINK employ-box
                        TO OPTIONAL depends-box
                        VIA empnum)
ADD APPL dept-detail
GENERATE APPL dept-detail
```

Upon execution, the "dept-detail" application will display the screen shown in Figure 4-13.

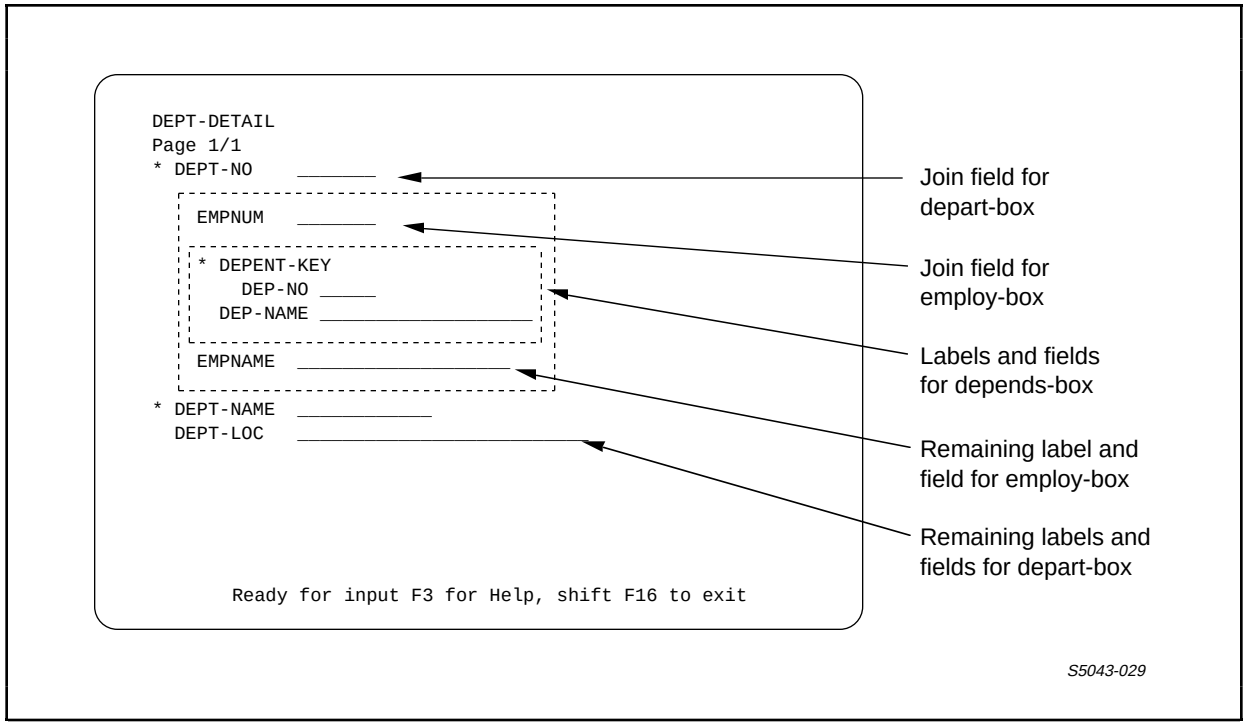


Figure 4-13. Sample Terminal Screen With Three Boxes

Example of the TREE Attribute With Four Boxes. When you build a tree structure for four boxes, you must identify the links that can exist between the boxes and determine the level at which each box is to reside in the tree structure.

Suppose, for example, that you want to generate an application that displays information about departments, the employees that belong to those departments, the dependents of those employees, and the type of benefit coverage that the employees have. This application must use the boxes described earlier in this section: "depart-box," "employ-box," and "depends-box." In addition, the application must use "benefits-box," a box that represents a file containing benefit-coverage information for each employee.

Figure 4-14 shows the part of the record descriptions associated with these boxes and also shows the partial contents of the files that these boxes represent.

Since the purpose of this application is to display information about departments, "depart-box" should be at the first level of the tree structure. "Employ-box" must be at the second level of the tree because this is the only box used by the application that can be linked to "depart-box." (The other boxes do not have appropriate join fields.)

Both "depends-box" and "benefits-box" have join fields that would allow you to link them either to "employ-box" or to each other. Since an employee might have benefits coverage without having any dependents or might have dependents without having benefits coverage, the satisfactory solution is to make "employ-box" the parent box of both "depends-box" and "benefits-box." This solution will allow the generated application to read or insert either a dependent or benefit record as long as a matching employee record exists.

Because "employ-box," which is to reside at the second level of the tree, is the parent of both "depends-box" and "benefits-box," both of these boxes must reside at the third level of the tree.

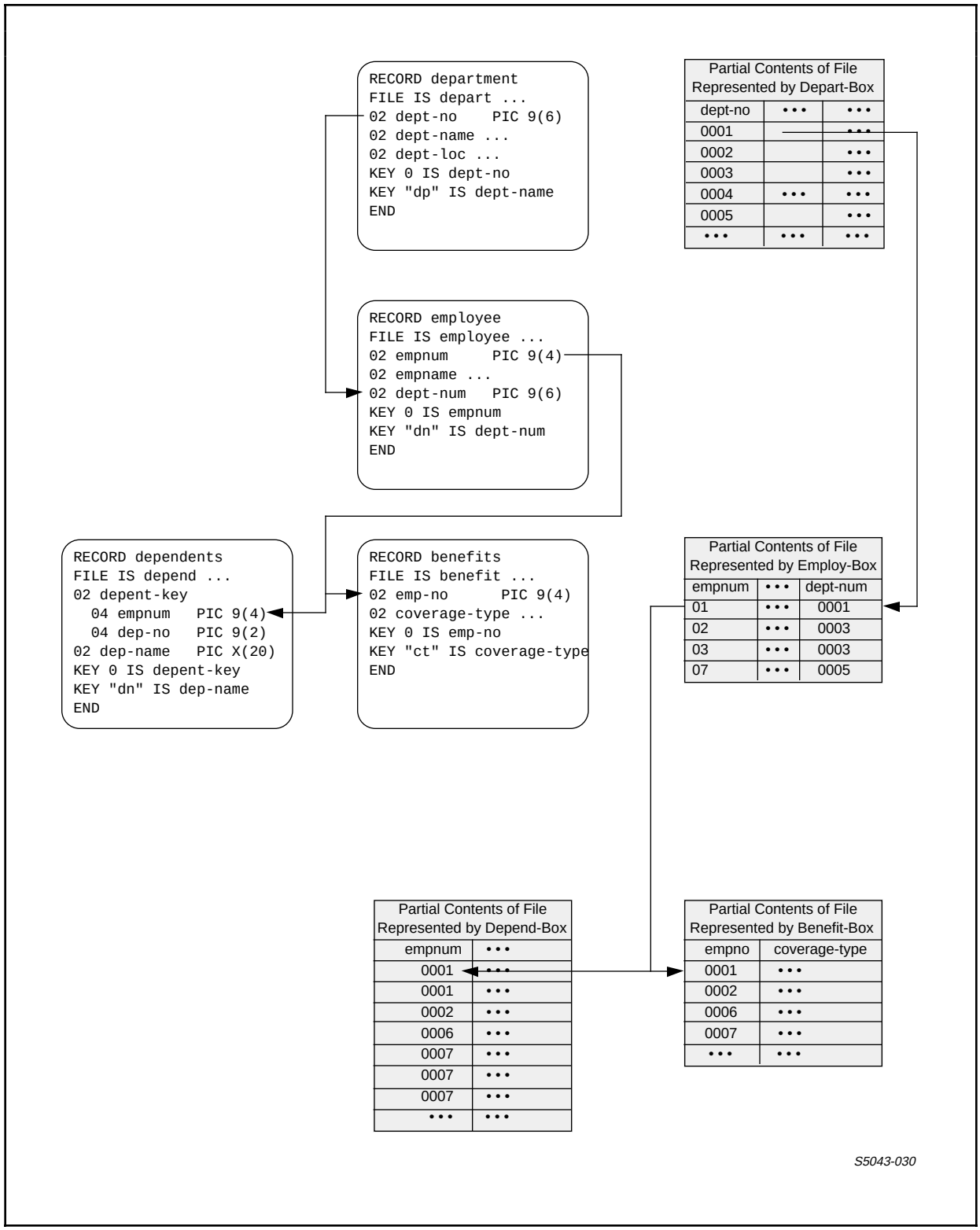


Figure 4-14. Sample Record Descriptions and File Contents for Four Boxes

To build this tree structure for an application, you set a value for the TREE attribute as follows:

1. Identify "depart-box" as being at the first level of the tree structure:

```
SET APPL TREE (02 depart-box
```

2. Identify "employ-box" as being at the second level of the tree structure:

```
SET APPL TREE (02 depart-box
                  04 employ-box
```

3. Provide the LINK OPTIONAL that connects "employ-box" to its parent "depart-box":

```
LINK dept-no TO OPTIONAL dept-num
        |                    |
        |              join field of "employ-box"
        |
   join field of "depart-box"
```

4. Include the LINK OPTIONAL with the part of the TREE value that identifies a level for "employ-box":

```
SET APPL TREE (02 depart-box
                  04 employ-box
                     LINK dept-no TO OPTIONAL dept-num
```

5. Identify "depends-box" as being at the third level of the tree:

```
SET APPL TREE (02 depart-box
                  04 employ-box
                     LINK dept-no TO OPTIONAL dept-num
                     06 depends-box
```

6. Provide the LINK OPTIONAL that links "depends-box" to its parent "employ-box":

```
LINK employ-box TO OPTIONAL depends-box VIA empnum
         |                          |         |
         |                          |     Name of join
         |                          |    field for both
         |                          |       boxes
         |                          |
         |                 Name of child box
Name of parent box
```

7. Include the LINK OPTIONAL in Step 6 with the part of the TREE value that defines a level for "depends-box":

```
SET APPL TREE (02 depart-box
                  04 employ-box
                     LINK dept-no TO OPTIONAL dept-num
                     06 depends-box
                        LINK employ-box
                        TO OPTIONAL depends-box
                        VIA empnum
```

8. Identify the level of "benefits-box":

```
SET APPL TREE (02 depart-box
                  04 employ-box
                     LINK dept-no TO OPTIONAL dept-num
                     06 depends-box
                        LINK employ-box
                        TO OPTIONAL depends-box
                        VIA empnum
                     06 benefits-box
```

9. Provide the LINK OPTIONAL that connects "benefits-box" to "employ-box":

```
LINK empnum TO OPTIONAL emp-no
        |                 |
        |            join field of "benefits-box"
        |
    join field of "employ-box"
```

10. Include the LINK OPTIONAL with the part of the TREE value that identifies the level of "benefits-box":

```
SET APPL TREE (02 depart-box
                  04 employ-box
                     LINK dept-no TO OPTIONAL dept-num
                     06 depends-box
                        LINK employ-box
                        TO OPTIONAL depends-box
                        VIA empnum
                     06 benefits-box
                        LINK empnum TO OPTIONAL emp-no)
```

The preceding TREE attribute builds the tree structure shown in Figure 4-15.

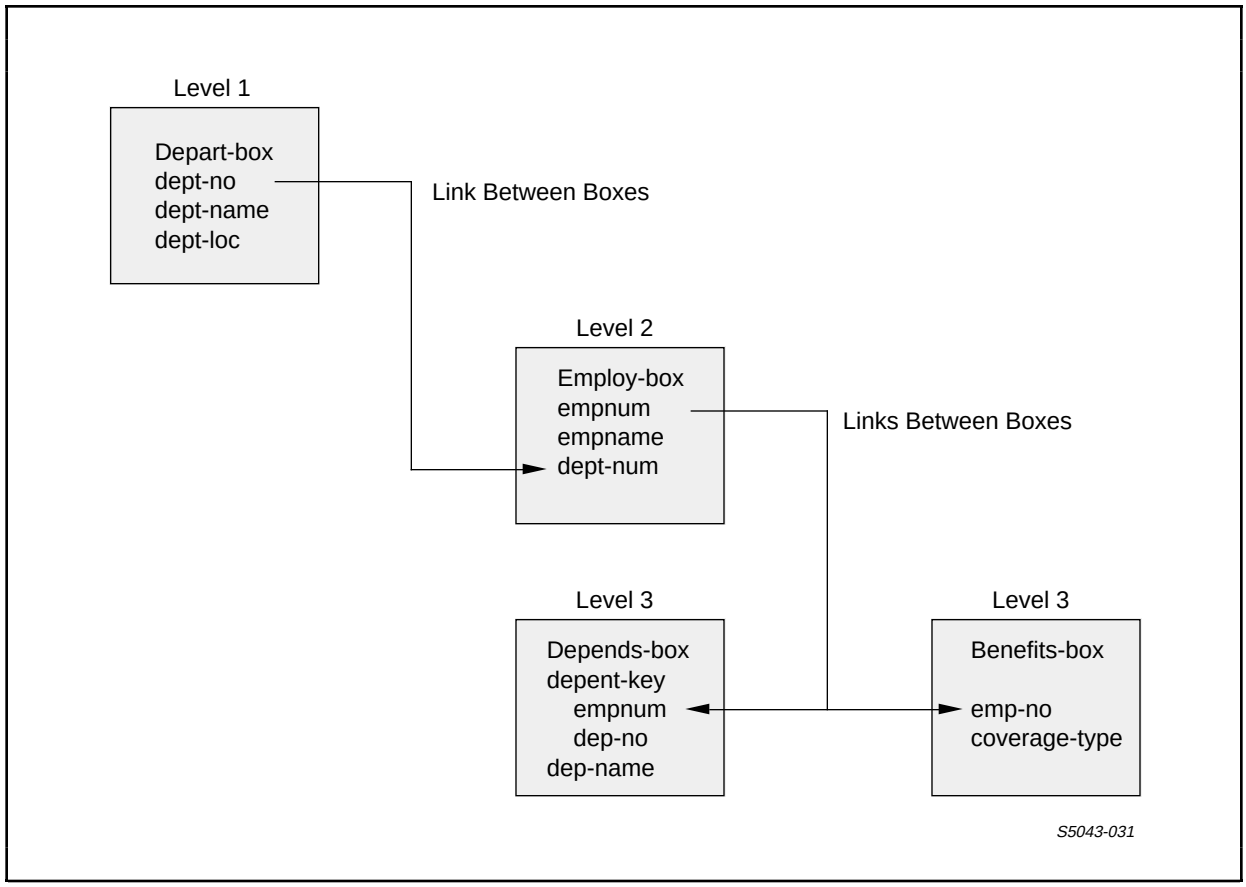


Figure 4-15. Sample Tree Structure With Four Boxes

You could generate an application that uses the tree structure shown in Figure 4-15 by entering the following ENABLE commands:

```
ADD BOX depart-box, RECORD department
ADD BOX employ-box, RECORD employee
ADD BOX depends-box, RECORD dependents
ADD BOX benefits-box, RECORD benefits
SET APPL PATHCOMFILE p7
SET APPL TREE (02 depart-box
                  04 employ-box
                     LINK dept-no TO OPTIONAL dept-num
                     06 depends-box
                        LINK employ-box
                        TO OPTIONAL depends-box
                        VIA empnum
                     06 benefits-box
                        LINK empnum TO OPTIONAL emp-no)
ADD APPL dept-benefits
GENERATE APPL dept-benefits
```

Upon execution, this application would display the terminal screen shown in Figure 4-16.

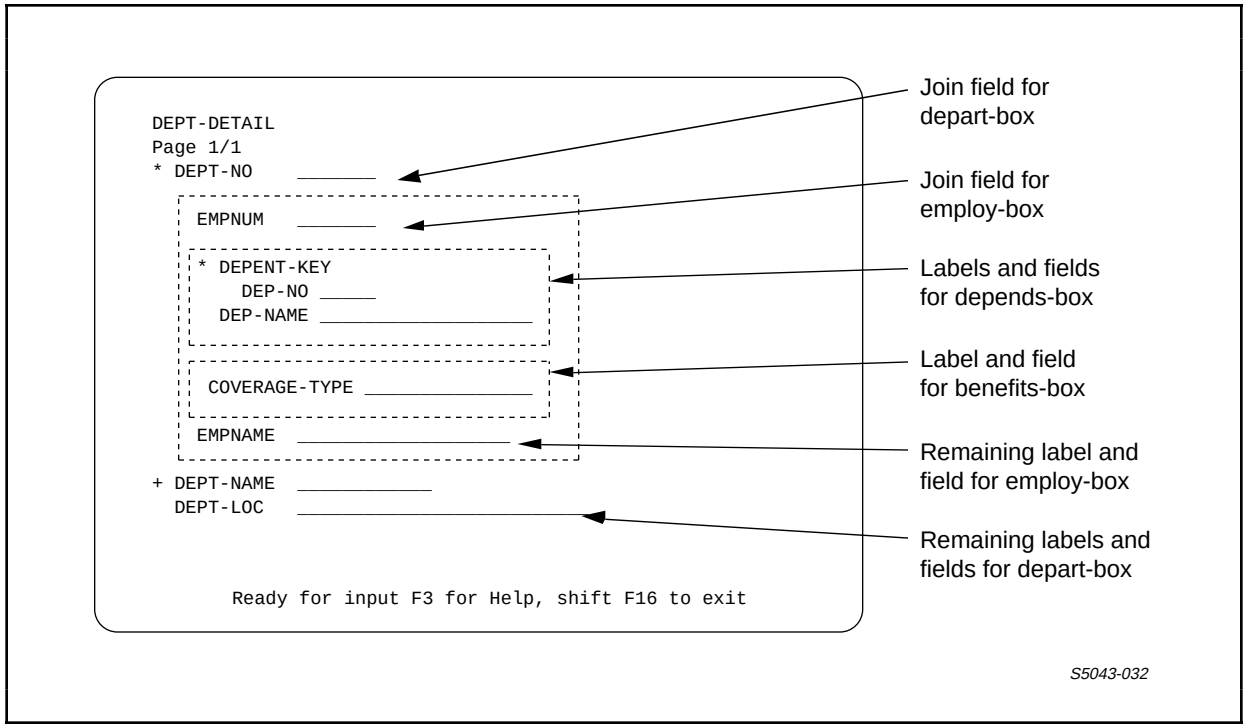


Figure 4-16. Sample Terminal Screen With Four Boxes

More TREE Attribute Examples. The following TREE attribute associates a box with a single-file application:

```
SET APPL TREE (02 employee)
```

Consider the following TREE attribute, which identifies three boxes for a multifile application: "odetail," "order," and "parts." The "odetail" box is at the highest level of the tree and the "order" and "parts" boxes are at the next level:

```
SET APPL TREE (01 odetail
                  02 order LINK odetail TO order VIA ordernum
                  02 parts LINK odetail TO parts VIA partnum)
```

For boxes named "odetail," "order," "seller," "parts," "fromsup," and "supplier," the links are as follows:

- "Odetail" is linked to "parts" with "partnum" as the join field for both boxes.
- "Order" is linked to "odetail" with "ordernum" as the join field of both boxes.
- "Order" is linked to "seller." "Salesman" is the join field for the "order" box, and "empnum" is the join field for the "seller" box.
- "Parts" is linked to "fromsup" with "partnum" as the join field for both boxes.
- "Fromsup" is linked to "supplier" with "suppnum" as the join field for both boxes.

The following TREE attribute identifies the level number of each box and the join field that links each box to a higher level box.

```
SET APPL TREE (01 odetail
                  02 order LINK odetail TO OPTIONAL order
                     VIA ordernum
                     03 seller
                        LINK salesman TO OPTIONAL empnum
                  02 parts LINK odetail TO OPTIONAL parts
                     VIA partnum
                     03 fromsup
                        LINK parts TO OPTIONAL fromsup
                        VIA partnum
                           04 supplier
                              LINK fromsup
                              TO OPTIONAL supplier
                              VIA suppnum)
```

Alternatively, you could use the following TREE attribute to build a tree structure for the same boxes:

```
SET APPL TREE (01 order
                   02 seller LINK salesman TO OPTIONAL empnum
                   02 odetail
                      LINK order TO OPTIONAL odetail
                      VIA ordernum
                      04 parts
                         LINK odetail TO OPTIONAL parts
                         VIA partnum
                         06 fromsup
                            LINK parts TO OPTIONAL fromsup
                            VIA partnum
                            08 supplier
                               LINK fromsup
                               TO OPTIONAL supplier
                               VIA suppnum)
```

UPDATE Attribute

The UPDATE attribute indicates whether the application can, or cannot, update records in the data base file represented by the box. The syntax of the UPDATE attribute is:

```
UPDATE { ON  }
       { OFF }

ON

  indicates that the application can update records in the
  data base file.

OFF

  indicates that the application cannot update records in
  the data base file.

The starting value of the UPDATE attribute is ON.
```

Consideration

If you supply ON as a value for the UPDATE attribute, the value of the READ attribute must also be ON. ENABLE imposes this requirement because an application must read a record before updating it.

Example

The following command sets the UPDATE box attribute to OFF:

```
SET BOX UPDATE OFF
```

VALUES Attribute

The VALUES attribute indicates that the application is, or is not, to display initial values from the record description on the base screen (the screen that appears when you first execute the application) within a box. The syntax of the VALUES attribute is:

```
VALUES { ON  }
       { OFF }

ON

  indicates that the application displays any initial
  values from DDL VALUE clauses in the record description.

OFF

  indicates that initial values are to be zeros and spaces.

The starting value of the VALUES attribute is OFF.
```

Consideration

If VALUES is ON, the application displays zeros or blanks (whichever is appropriate) for any field or group in the record description that does not have a DDL VALUE clause.

Example

The following command sets the VALUES box attribute to ON:

```
SET BOX VALUES ON
```

SECTION 5

OPERATING COMMANDS

The ENABLE operating commands perform the following functions:

- Stop ENABLE (EXIT)
- Display the syntax of the ENABLE commands (HELP)
- Direct the output listing to a specified file (OUT)
- Edit or repeat a command line (FC)
- Display the current setting of the environmental parameters (ENV)
- Enter commands from a specified file (OBEY)
- Set the default system, volume, and subvolume for expansion of file names (CMDSYS, CMDVOL, OBEYSYS, OBEYVOL, SYSTEM, and VOLUME)

This section describes the operating commands in alphabetical order. Because many of the operating commands affect the setting of the default system, volume, and subvolume that ENABLE uses to expand file names, a brief discussion of file names and file name expansion precedes the description of these commands.

FILE NAME EXPANSION

To refer to Tandem files, use either "local," or "network" file names.

Local File Names

The local form identifies files on a single Tandem system. If the file is not in your current subvolume or volume, you must qualify the name with a specific subvolume or volume. The syntax for local file names is:

[$<volume-name>.][<subvolume>.]<disc-file-name>

<volume-name>	identifies a physical disc pack mounted on a disc drive. This name may be up to seven characters long.
<subvolume-name>	identifies a set of files on a disc, as defined by the user. This name may be up to eight characters long.
<disc-file-name>	identifies a particular file within the subvolume. This name may be up to eight characters long.

The following example illustrates a local file name:

```
$mkt.sales.tracking
```

"$mkt" is the volume name.

"sales" is the subvolume name.

"tracking" is the disc file name.

Network File Names

The network form identifies files on remote systems. The syntax for network file names is:

\<system-name>.$<remote-volume>.<subvolume>.<disc-file-name>

<system-name>	identifies a specific system within a network. This name may be up to seven characters long.
<remote-volume>	identifies a physical disc pack mounted on a disc drive. This name may be up to six characters long.
<subvolume-name>	identifies a set of files on a disc, as defined by the user. This name may be up to eight characters long.
<disc-file-name>	identifies a particular file within the subvolume. This name may be up to eight characters long.

The next example illustrates the network form of the file name presented in the previous example:

```
\nusys.$mkt.sales.tracking
```

In this case:

"\nusys"	is the name of the system.
"$mkt"	is the remote volume name.
"sales"	is the subvolume name.
"tracking"	is the disc file name.

The file system identifies devices (such as tape drives or printers) in a similar manner; for example:

```
\tsb.$tape1
```

specifies the tape drive "$tape1" on system "\tsb."

Although the file system uses expanded file names internally, you can specify a partial name when entering commands. At a minimum, you must supply the <disc-file-name> to identify a file on the default system, volume and subvolume associated with your command interpreter. The file system uses these defaults to expand partial names either to the network form or the local form, as required.

ENABLE also expands names according to the defaults mentioned above. Initially, ENABLE uses the defaults associated with the command interpreter to expand file and device names. You can alter these default settings using ENABLE operating commands.

CMDSYS Command

The CMDSYS command sets the default system for expansion of file names other than obey file names. The syntax of the CMDSYS command is:

```
CMDSYS [ \<system-name> ]

<system-name>

  is a GUARDIAN system name.

The initial CMDSYS setting is the system in effect when you
start ENABLE.
```

Considerations

To set the default system name to the local system, omit the <system-name> parameter. When this parameter is omitted, ENABLE uses the local form for file expansion. Use of the local form allows access to all volume and device names on the system.

When you supply a <system-name> (even if the name of the local system is specified), ENABLE uses the network form for file name expansion. Use of the network form makes only volume or device names having six characters or less accessible.

If <system-name> is either invalid or would be invalid when combined with the current CMDVOL setting, ENABLE displays an error message and does not change the current CMDSYS setting.

Example

The following example idendifies "\ny" as the system name for expansion of file names other than obey files:

```
CMDSYS \ny
```

CMDVOL Command

The CMDVOL command sets the default volume and subvolume names for expansion of file names other than obey file names. The syntax of the CMDVOL command is:

```
CMDVOL { $<volume>              }
       { $<volume>.<subvolume> }
       { <subvolume>            }

<volume>

  is a GUARDIAN volume name.

<subvolume>

  is a GUARDIAN subvolume name.

The initial CMDVOL setting is the volume and subvolume in
effect when you start ENABLE.
```

Considerations

If you omit $<volume>, ENABLE uses the current volume setting.

If you omit <subvolume>, ENABLE uses the current subvolume setting.

If you omit both, ENABLE uses the volume and subvolume in effect when you start ENABLE.

ENABLE displays an error message and does not change the default CMDVOL setting under the following conditions:

- Either $<volume> or <subvolume> is an invalid name.
- Either $<volume> or <subvolume> is invalid when expanded using the current defaults.

Examples

The following are examples of the CMDVOL command:

```
CMDVOL $mkt.abc  <-  sets the default volume and subvolume
CMDVOL $engr  <----  sets the default volume
CMDVOL zot  <------  sets the default subvolume
```

ENV Command

The ENV command displays the current settings of program environment parameters. The syntax of the ENV command is:

Consideration

If you enter a parameter that is not valid, ENABLE ignores the ENV command and displays a warning message. If you do not enter a parameter with the ENV command, ENABLE displays the values for all the environmental parameters.

Examples

To display the current setting of the OBEYSYS parameter, enter:

```
ENV OBEYSYS
```

To display the current settings for all environmental parameters, enter:

```
ENV
```

EXIT Command

The EXIT command stops ENABLE. The syntax of the EXIT command is:

```
EXIT
```

FC Command

The FC command lets you edit or repeat a command line. The syntax of the FC command is:

```
FC
```

When this command is used, it displays up to 132 characters of the previous command line and then prompts for input with a period (.).

To repeat the previous command, press RETURN after the period prompt (.).

To edit the command, enter the subcommands described below, then press RETURN.

FC accepts three subcommands:

R<replacement-string>	Replace one or more characters
I<insertion-string>	Insert one or more characters
D	Delete one character

Replacement, insertion, and deletion in the displayed command line begin with the character positioned directly above the subcommand (R, I, or D) you enter.

Subcommand R replaces characters in the command line with <replacement-string> on a one-for-one basis. Subcommand I inserts the characters of <insertion-string> in the command line. Subcommand D deletes the character above it in the command line; you can repeat the D subcommand for each contiguous character to be deleted. If you enter a string without a preceding subcommand, R is assumed.

If you want to specify more than one subcommand on a line, you can separate the subcommands by entering a double slash. For example:

```
%set boz record eemployee
.      x//      d  <----  The character "z" is replaced by
                          "x," since replacement is the
                          default subcommand.  The extra "e"
                          is deleted
```

After you edit the line, FC displays the modified command line and prompts for another subcommand. FC terminates when you enter only a carriage return in response to the prompt. ENABLE then attempts to execute the new version of the command line.

If you do not want to execute the edited command line, you can abort the FC command in any of the following ways:

- Press the BREAK key.
- Press the CTRL and Y keys simultaneously.
- Enter a double slash (//) in columns 1 and 2, immediately followed by a carriage return.

Example

This example shows the use of the FC command to edit several errors in an ENABLE command line:

```
:ENABLE

%set boz eemployee  <-----  A command is miskeyed.
       ^
  Invalid syntax  <-------  ENABLE displays an error
                            message.

%FC  <--------------------  The FC command is entered.

%set boz eemployee  <-----  FC prompts for editing.
.        d  <-------------  The extra letter "e" is marked
                            for deletion.

%set boz employee
.        irecord  <-------  The string "record" is inserted.
```

```
%set boz recordemployee
.              i   <------
```

A blank is inserted by pressing the space bar following the I subcommand.

```
%set boz record employee
.      rx  <--------------
```

The letter "z" is replaced by the letter x.

```
%set box record employee
.  <----------------------
```

The RETURN key indicates that editing is complete. ENABLE accepts the edited command.

HELP Command

The HELP command displays the syntax of ENABLE commands. The syntax of the HELP command is:

```
HELP [ <command-name>      ]
     [ {<}<symbol-name>{>} ]

<command-name>

  is the name of an ENABLE command for which the syntax is
  to be displayed.

<symbol-name>

  is any symbol that is enclosed in angle brackets when
  displayed by the <command-name> option. You must include
  the angle brackets with <symbol-name>.
```

Consideration

If you omit the parameters, the HELP command displays the names of all ENABLE commands.

Examples

To display the syntax of all ENABLE commands, enter:

```
HELP
```

To display the syntax of the GENERATE command, enter:

```
HELP generate
```

To display information about valid terminal types, enter:

```
HELP <terminal-type>       (brackets included)
```

OBEY Command

The OBEY command causes commands to be read from a specified file. The syntax of the OBEY command is:

```
OBEY <filename>

<filename>
   is a Tandem file name.
```

Considerations

If you do not supply a complete file name, ENABLE expands the obey file name with the default OBEYSYS and OBEYVOL settings.

When you enter the OBEY command, ENABLE reads and processes the commands from the named file until it encounters an end-of-file. At end-of-file, ENABLE closes the obey file and accepts command input from the file (or terminal) from which the OBEY command was read. You can specify additional OBEY commands within an obey file. Obey files can be nested to a depth of four.

ENABLE displays an error message and continues to accept command input from the file from which the OBEY command was entered if:

- The obey command is invalid
- The obey file does not exist
- The obey file cannot be opened

If ENABLE detects an error during processing of commands from an obey file, it:

- Issues an error message
- Terminates processing from the obey file (and any other obey files currently open in the case of nested obey files)
- Closes the obey file and any other obey files currently open

If the original input file for ENABLE was a terminal, ENABLE issues a prompt to the terminal. If the input file was not a terminal, ENABLE terminates.

Example

To tell ENABLE to read commands from a file named "enabcom" on a subvolume named "abc," enter:

```
OBEY abc.enabcom
```

OBEYSYS Command

The OBEYSYS command sets the default system for expansion of obey file names. The syntax of the OBEYSYS command is:

```
OBEYSYS [ \<system-name> ]

<system-name>

  is a GUARDIAN system name.

The initial OBEYSYS setting is the system in effect when
you start ENABLE.
```

Considerations

To set the default system name to the local system, omit the <system-name> parameter. When this parameter is omitted, ENABLE uses the local form for file name expansion. Use of the local form allows access to all volume and device names on the system.

When you supply a <system-name> (even if the name of the local system is specified), ENABLE uses the network form for file name expansion. Use of the network form makes only volume, or device names having six characters or less accessible.

If <system-name> is either an invalid system name or would be an invalid system name when combined with the current OBEYVOL setting, ENABLE displays an error message and does not change the OBEYSYS setting.

Example

The following OBEYSYS command identifies a system named "\sd."

```
OBEYSYS \sd
```

OBEYVOL Command

The OBEYVOL command sets the default volume and subvolume for expansion of obey file names. The syntax of the OBEYVOL command is:

```
OBEYVOL { $<volume>              }
        { $<volume>.<subvolume> }
        { <subvolume>            }

<volume>

  is a GUARDIAN volume name.

<subvolume>

  is a GUARDIAN subvolume name.

The initial OBEYVOL setting is the volume and subvolume in
effect when you start ENABLE.
```

Considerations

If you omit $<volume>, ENABLE uses the current volume setting.

If you omit <subvolume>, ENABLE uses the current subvolume setting.

If you omit both, and the file from which the OBEYVOL command was read is not a disc file, ENABLE displays an error message and the OBEYVOL setting remains unchanged.

ENABLE displays an error message and does not change the default OBEYVOL setting under the following conditions:

- Either $<volume> or <subvolume> is an invalid name.
- Either $<volume> or <subvolume> is invalid when expanded using the current defaults.

Example

The following OBEYVOL command identifies "$mkt.abc" as the default volume and subvolume with which obey file names are to be expanded:

```
OBEYVOL $mkt.abc
```

OUT Command

The OUT command directs the output listing to a specified file. The syntax of the OUT command is:

```
OUT <file-name>

  or

<command> /OUT <file-name> / <parameter>

<file-name>

  is a Tandem file name.

<command>

  is an ENABLE command or an ENABLE Operating command.

<parameter>

  is a valid parameter for that command.
```

Considerations

The first form of the OUT command causes permanent redirection of the output.

The second form of the OUT command causes temporary redirection of the output. To use this form of the OUT command, insert the OUT command between <command> and the first valid <parameter> for that command; for example:

```
HELP/OUT helpfile/GENERATE
```

If <file-name> has the form of a disc file name and the disc file does not exist, ENABLE creates an edit-type file. If <file-name> is an existing disc file, ENABLE appends the output to the file.

ENABLE displays an error message and does not redirect the listing under the following conditions:

- <file-name> is invalid.

- ENABLE cannot open <file-name>.

Examples

The following OUT command permanently directs the ENABLE output listing to the file "enabout."

```
OUT enabout
```

In the following example, output is temporarily directed to the spooler:

```
HELP/OUT $s.#hold/GENERATE
```

SYSTEM Command

The SYSTEM command sets the default system for expansion of all file names. The syntax of the SYSTEM command is:

```
SYSTEM [ \<system-name> ]

<system-name>

  is a GUARDIAN system name.

The initial SYSTEM setting is the system in effect when you
start ENABLE.
```

Considerations

To set the default system name to the local system, omit the <system-name> parameter. When this parameter is omitted, ENABLE uses the local form for file name expansion. Use of the local form allows access to all volume and device names on the system.

When you supply a <system-name> (even if the name of the local system is specified), ENABLE uses the network form for file name expansion. Use of the network form makes only volume or device names having six characters or less accessible.

If <system-name> is either an invalid system name or would be an invalid system name when combined with the current VOLUME setting, ENABLE displays an error message and does not change the SYSTEM setting.

Example

The following SYSTEM command identifies "\ny" as the system to be used for expansion of file names:

```
SYSTEM \ny
```

VOLUME Command

The VOLUME command sets the default volume and subvolume for expansion of all file names. The syntax of the VOLUME command is:

```
VOLUME { $<volume>              }
       { $<volume>.<subvolume> }
       { <subvolume>            }

<volume>

  is a GUARDIAN volume name.

<subvolume>

  is a GUARDIAN subvolume name.

The initial VOLUME setting is the volume and subvolume in
effect when you start ENABLE.
```

Considerations

If you omit $<volume>, ENABLE uses the current volume setting.

If you omit <subvolume>, ENABLE uses the current subvolume setting.

If you omit both, and the file from which the OBEYVOL command was read is not a disc file, ENABLE displays an error message and the VOLUME setting remains unchanged.

A VOLUME command is equivalent to entering both an OBEYVOL and a CMDVOL command with the same specification. Note that this does not necessarily mean that the settings for OBEYVOL and CMDVOL would be identical following a VOLUME command. If, for example, the default value for OBEYVOL is:

```
$v1.a
```

and the default value for CMDVOL is:

```
$v2.b
```

A subsequent command of "VOLUME x" yields:

$v1.x for OBEYVOL, and

$v2.x for CMDVOL.

ENABLE displays an error message and does not change the default OBEYVOL setting under the following conditions:

- Either $<volume> or <subvolume> is an invalid name.
- Either $<volume> or <subvolume> is invalid when expanded using the current defaults.

Example

The following command identifies "$mkt" as the default volume for expansion of file names:

VOLUME $mkt

SECTION 6

APPLICATION EXECUTION

After you generate an application, you must establish a PATHWAY system to execute the application. To establish a PATHWAY system, you enter commands that:

- Create a PATHWAY Monitor (PATHMON) process, which is the central controlling process of a PATHWAY system
- Create a PATHCOM process. PATHCOM is the command interface to PATHMON
- Configure the PATHWAY system using the PATHCOM command file created by ENABLE

You can place these commands in an obey file for convenient definition, execution, and termination of the PATHWAY system. To create such an obey file, enter the commands described in Figure 6-1 in an edit-type file.

```
PURGE <log-file>, <path-control-file>
CREATE <log-file>
ASSIGN PATHCTL, <path-control-file>
PATHMON/NAME <pathmon-name>, NOWAIT, CPU 0, OUT <log-file>/
PATHCOM/IN <pathcom-command-file-name>/ <pathmon-name>
PATHCOM <pathmon-name>;RUN <program-name>
PATHCOM <pathmon-name>; SHUTDOWN, WAIT

<log-file>

  is a file to which PATHMON can report errors and changes
  in status.

<path-control-file>

  is a disc file to which the PATHCTL file is assigned.
  PATHMON uses the PATHCTL file to maintain status
  information and to store the PATHWAY system
  configuration.

<pathmon-name>

  is the name of a PATHMON process.  On a local system,
  <pathmon-name> must:

  • Consist of no more than six characters

  • Begin with a dollar sign ($)

  • Be unique among the names of the processes executing
    on the system

  If you use the PATHWAY system across a network,
  <pathmon-name> must:

  • Consist of no more than five characters

  • Begin with a dollar sign ($)

  • Be unique among the names of the processes executing
    on both systems
```

Figure 6-1. Commands to Establish a PATHWAY System and Execute an Application (Continued next page)

<pathcom-command-file-name> is the name of the PATHCOM command file created by ENABLE. ENABLE creates this file if you supply a value for the PATHCOMFILE attribute when you generate an application. <program-name> is the name of the generated application. Within the PATHMON command, CPU 0 identifies the processor in which the primary PATHMON process runs. The PATHCOM command file produced by ENABLE identifies CPU 1 as the processor in which the backup process runs. Your system manager may want you to select different CPUs by editing this command and the PATHCOM command file.

Figure 6-1. Commands to Establish a PATHWAY System and Execute an Application (Continued)

The commands listed in Figure 6-1 do the following:

PURGE <log-file>, <path-control-file>	Purges the log file and the path-control-file.
CREATE <log-file>	Creates a log file.
ASSIGN PATHCTL, <path-control-file>	Assigns PATHCTL (the default path-control-file) to the file of your choice.
PATHMON/NAME <pathmon-name>, NOWAIT, CPU 0, OUT <log-file>/	Starts a PATHMON process and gives the process a name. This command also identifies the primary CPU in which the PATHMON process is to run and identifies the log file used by the process.
PATHCOM/ IN <pathcom-command-file/ <pathmon-name>	Configures the PATHWAY system.

PATHCOM <pathmon-name>; RUN <program-name>	Executes an application.
PATHCOM <pathmon-name>; SHUTDOWN, WAIT	Stops the PATHWAY system.

Figure 6-2 shows a sample obey file that uses these commands.

```
PURGE log1, enabctl
CREATE log1
ASSIGN PATHCTL, enabctl
PATHMON/ NAME $one, NOWAIT, CPU 0, OUT log1/
PATHCOM/ IN enabpath/$one
PATHCOM $one;RUN employee-prog
PATHCOM $one; SHUTDOWN, WAIT
```

Figure 6-2. Sample Obey File

EXECUTING THE OBEY FILE

To execute an obey file, enter the following in response to the command interpreter prompt:

```
OBEY <file-name>
```

where <file-name> is the name of the obey file.

STARTING A PATHWAY SYSTEM AFTER A SHUTDOWN

When you execute a SHUTDOWN command (the last command in Figure 6-1), this command brings the PATHWAY system to an idle state. PATHMON writes the internal configuration information to the assigned PATHCTL file and then stops.

To avoid the time involved in configuring the PATHWAY system, you can start PATHWAY in a cool state by entering the commands shown in Figure 6-3. A cool state is one in which a previous PATHWAY configuration exists. When you start PATHWAY in a cool state, PATHMON uses the configuration information stored in the existing assigned PATHCTL file.

```
ASSIGN PATHCTL, <path-control-file>
PATHMON/NAME <pathmon-name>, CPU 0, NOWAIT, OUT <log-file> /

PATHCOM <pathmon-name>; START PATHWAY COOL

<path-control-file>

  is the file to which you assigned the PATHCTL file when
  you originally established the PATHWAY system.

<pathmon-name>

  is a name that you supply for the PATHMON process. This
  name must conform to the rules described in Figure 6-1.

<log-file>

  is the name of the log file.
```

Figure 6-3. Starting PATHWAY in a Cool State

ESTABLISHING A PATHWAY SYSTEM FOR TWO OR MORE USERS

The PATHCOM command file generated by ENABLE contains commands that allow up to five individuals to use a PATHWAY system concurrently. If you want to allow several individuals to use an application at the same time, you must create separate obey files to establish the PATHWAY system, execute the application, and SHUTDOWN the PATHWAY system.

Figure 6-4 shows an example of an obey file that establishes a PATHWAY system. Note that this figure contains the first five lines from Figure 6-2.

```
PURGE log1, enabctl
CREATE log1
ASSIGN PATHCTL, enabctl
PATHMON/ NAME $one, NOWAIT, CPU 0, OUT log1/
PATHCOM/ IN enabpath/$one
```

Figure 6-4. Sample Obey File That Establishes a PATHWAY System

If you execute such an obey file, the PATHWAY system remains running until you execute a command to SHUTDOWN the system. Since an idle PATHWAY system consumes few system resources, in most cases you can leave your PATHWAY system running as long as you like.

To allow individuals to execute the application, you can create an obey file such as the one shown in Figure 6-5. This obey file contains the PATHCOM command that causes execution of an application.

```
PATHCOM $one; RUN employee-prog
```

Figure 6-5. Sample Obey File That Executes an Application

To SHUTDOWN the PATHWAY system, create an obey file similar to the one shown in Figure 6-6. The PATHCOM command shown in this figure brings the PATHWAY system to an idle state and stops PATHMON.

```
PATHCOM $one; SHUTDOWN, WAIT
```

Figure 6-6. Sample Obey File That Stops a PATHWAY System

SECTION 7

ENABLE SCREENS

Standard ENABLE applications display the following terminal screens:

- A screen through which you can retrieve or update records stored in data base files
- A screen that displays text that briefly describes the way you use an ENABLE application

STANDARD SCREEN FORMATS

The following paragraphs describe the format of the standard screens through which you can retrieve and update records using both single-file and multifile applications. The discussion assumes that both types of application have been generated using the starting values of the attributes that affect screen format. These attributes are:

BOXTITLE 1
BOXTITLE 2
BOXTITLE 3
: The values of these attributes indicate whether text appears with a box on the screen. The starting values of these attributes are null, meaning no text appears with a box.

EXCLUDE
: The value of this attribute identifies fields from a record description that the application does not display on the screen. The starting value of this attribute is null, meaning the application displays all fields from a record description.

HEADINGS — The value of this attribute indicates the type of labels, if any, that the application uses to identify fields on the screen. The starting value of this attribute is DDLFIELDNAMES, meaning the application displays field names from the record description as screen labels.

INCLUDE — The value of this attribute identifies the order in which the application displays screen field and label pairs. If you supply a value for this attribute but omit one or more fields from a record description, the application does not display the omitted field. The starting value of this attribute is null, meaning all fields that are in the record description appear on the screen in the same order that they appear in the record description.

SCREENFORMAT — The value of this attribute identifies the screen layout that the application uses for screen label and field pairs. The starting value of this attribute is UNCOMPRESSED, meaning the application displays at most one screen label and field pair on a screen line.

SIZE — The value of this attribute identifies the number of records that the application can display within a box. The starting value of this attribute is 1.

TITLE — The value of this attribute identifies the text that the application displays as a screen title. The default value of this attribute is the application name.

VALUES — The value of this attribute identifies the type of initial values that the application displays for screen fields. If VALUES is ON, the application displays initial values from a record description. The starting value of this attribute is OFF, meaning that the application displays zeros or blanks as initial values in screen fields.

Standard Screen Format for a Single-File Application

Figure 7-1 shows a record description and a series of ENABLE commands used to generate a single-file application.

```
Record Description:

  RECORD employee.
  FILE IS employee  KEY-SEQUENCED.
  02 empnum         PIC 9(4).
  02 empname        PIC X(18).
  02 dept.
     04 regnum      PIC 9(2).
     04 branchnum   PIC 9(2).
  02 job            PIC X(12).
  02 age            PIC 99.
  02 salary         PIC 9999V99.
  02 vacation       PIC 999.
  KEY 0 IS empnum.
  KEY "dp" IS dept.
  KEY "en" IS empname.
  END

ENABLE Commands:

  SET BOX RECORD employee
  ADD BOX employee
  SET APPL TREE (01 employee)
  SET APPL PATHCOMFILE enabpath
  ADD APPL employee-prog
  GENERATE employee-prog
```

Figure 7-1. Record Description and ENABLE Commands for a Sample Single-File Application

Figure 7-2 shows the format of the standard screen displayed by this sample application on a T16-651x, T16-652x, or T16-653x terminal. The line numbers that appear to the right of the figure do not actually appear on the screen; they refer to the explanatory notes that follow.

```
EMPLOYEE-PROG                                                    (1)
Page 1/1                                                         (2)
* EMPNUM      ____                                               (3)
+ EMPNAME     ____________________                               (4)
+ DEPT                                                           (5)
    REGNUM    __                                                 (6)
    BRANCHNUM __                                                 (7)
  JOB         __________                                         (8)
  AGE         __                                                 (9)
  SALARY      ____.00                                           (10)
  VACATION    ___                                               (11)
                                                                  .
                                                                  .
                                                                  .
    Ready for input   F3 for Help,shift F16 to Exit             (23)
                                                                 (24)
```

Line Number	Information Displayed by Application
(1)	The screen title of the application
(2)	The heading "Page 1/1" indicating that the current page is page 1 of a 1 page screen
(3)	The first screen label and field pair for an employee record: * EMPNUM ____ The asterisk (*) to the left of the label (EMPNUM) indicating that this field is a primary key field
(4)	The second screen label and field pair: + EMPNAME ____________________ The plus sign (+) to the left of the label (EMPNAME) indicating that this field is an alternate key field

Figure 7-2. Sample Standard Terminal Screen for a Single-File Application (Continued next page)

(5-7)	The screen label and field pairs for a group: + DEPT REGNUM __ BRANCHNUM __ Notice that: • A screen field does not appear with the group label (DEPT). • The elementary items (REGNUM and BRANCHNUM) of the group appear on the screen lines following the group label. The application indents two screen columns before displaying these elementary fields. • The plus sign (+) that appears to the left of the DEPT label indicates that this group is an alternate key field. The group is also a composite key field because it consists of more than one elementary field.
(6-11)	The remaining screen label and field pairs for an employee record
(23)	An informative message stating that the application is ready

Figure 7-2. Sample Standard Terminal Screen for a Single-File Application (Continued)

General Format of the Standard Screen

The following rules apply to the general format of a screen page:

- Line 1 displays the title of the application. If you do not provide a title, the program-unit-name of the application is displayed. This is the name you specify in an ADD APPL command.
- Line 2 displays the heading Page m/n, where m indicates the current screen page and n indicates the total number of screen pages.

Line 2 also displays a message when the application has performed a successful read operation. The message indicates the type of read operation performed and the name of the key field that was used. You can use this information to determine how the application will process a READ NEXT operation.

- Lines 3 through 20 display screen label and field pairs.
- Line 23 displays a message whenever the application successfully completes an operation.
- Line 24 displays a message if an error occurs during application execution. The message is highlighted and persists until you press an ENABLE function key.

  If this line displays a message immediately after the application executes but before any action can begin, an error that prevents the application from performing any operations has occurred. If this message appears, the application terminates as soon as you press any ENABLE function key.

  For T16-652x and T16-653x terminals, a message displayed on line 24 appears in DIM REVERSE video. For T16-651x terminals, a message on line 24 appears blinking.

  Appendix C lists the messages that might appear on this line.

Screen Label and Field Format

The following rules apply to the format of the label and field pairs displayed on a standard screen:

- Field names from a record description appear as labels.
- Fields declared as FILLER in a record description remain in the record but they do not appear on the screen and, therefore, you cannot update them. The application gives these fields null values if you perform an insert operation.
- Bracketed subscripts appear with the labels of fields declared with an OCCURS clause in a record description; for example:

```
COMMENT[1] _____________________________
COMMENT[2] _____________________________
```

- Each new level of a nested group field appears indented two screen columns. If the leftmost character of a label reaches screen column 30, the remaining group items begin in the same screen column as the label for which the leftmost character reached column 30.

- An asterisk (*) that appears to the left of a screen label identifies the primary key of a key-sequenced file and the courtesy key (the record number) of other file types.

- A plus sign (+) that appears to the left of a screen label identifies an alternate key field.

- When an application uses a relative, entry-sequenced, or unstructured file, the following label and field appear as the first screen pair of the record:

  * Record Number _____

  ENABLE supplies this label and field for the record number of such files. The screen section of the application defines this field as PIC Z(8)9 for entry-sequenced and unstructured files and as PIC -Z(8)9 for relative files.

- Up to 20 screen lines display screen pairs. A single screen pair does not break across screen pages. If all screen pairs for a record cannot fit on a single screen page, these pairs break as follows:

  --Groups appear on one page where possible. If a group must break across a screen page, up to five of the dominant group items repeat on the subsequent screen page before the first new item in the group is displayed.

  --Fields with OCCURS clauses appear on one page if possible.

- If a field cannot fit on a screen line, the field wraps around to the subsequent screen line. On the subsequent line, the wrapped field begins below in column one.

- For T16-652x and T16-653x terminals, the UNDERLINE attribute indicates the length of alphabetic fields, alphanumeric fields, and numeric fields without embedded decimal points. For examples of these fields, consider the following chart:

Field as Defined in a Record Description	Resulting Data Type	Field as Displayed on the Screen
02 sex PIC A.	Alphabetic	SEX _
02 location PIC X(3).	Alphanumeric	LOCATION ___
02 age PIC 99.	Numeric	AGE __
02 emp-no PIC 9(8) USAGE IS COMP.	Numeric	EMP-NO _____
02 vacation PIC S9(2).	Numeric	VACATION ___

For T16-651x terminals, underscore characters (_) indicate field length. The screen fields are similar to those shown in the preceding table.

For all terminals, the SCREEN SECTION defines numeric fields as PIC Z(n) and alphanumeric fields as PIC X.

- Numeric fields defined with embedded decimal points, such as PIC 9(4)V99, appear with the decimal and all following digits explicitly displayed as follows:

  SALARY ____.00

  The screen section of the application defines these fields as PIC Z(n).9(m).

Values Entered in Screen Fields

When you enter a value in a screen field, that value must conform to the data type of the screen field. Four broad categories of data types exist. They are:

- Alphabetic (PIC A) — You can enter letters of the alphabet and space characters in these fields.
- Alphanumeric (PIC X) — You can enter letters of the alphabet, digits, and special characters such as a hyphen in these fields.
- Numeric Integer (PIC 9) — You can enter digits and possibly a sign character in these fields.

- Numeric Noninteger (PIC 9 V) — You can enter digits and a decimal point character in these fields. When you enter digits, you must align them with the decimal point.

For all fields, you can enter up to the number of characters identified by the UNDERLINE attribute or underscore characters.

For the Record Number field of entry-sequenced, relative, or unstructured files, you can enter up to eight digits. When you enter a value in this field, you can use the field like a primary key and read the record whose record number matches the value you entered. You cannot, however, enter a value in a Record Number field and perform a generic read operation. (Refer to Section 8 for more information about this type of operation.)

For relative files only, you can enter a value in the Record Number field and perform an insert operation. The application ignores any value entered in this field for insert operations on entry-sequenced or unstructured files and assigns the next available number.

Values Returned to Screen Fields

If you perform a read operation, the application returns values from the data base file to the screen fields. If a read operation returns nonnumeric data for a field that was defined as numeric and if the application was generated with CHECKDATA ON, the application issues an error message and highlights the field with the invalid data.

If you perform a read operation on an entry-sequenced, relative, or unstructured file, the value of the Record Number field indicates the position of the record just read within the file. Because the file system considers zero as the number of the first record in files of these types, the Record Number field displays blanks when you read the first record in such a file.

Cursor Position

When the standard screen appears, the cursor is initially positioned at the left of the first screen field (which might be a record number field). The application accepts entries to screen fields and tabs to subsequent fields until you press a function key.

When you enter a value in the last screen field, the cursor returns to the first screen field. For applications generated for T16-652x, T16-653x, and IBM-327x terminals, the cursor does not automatically tab to the next screen field when positioned in a key field, the last subentry of a group key field, or the last field of a nested box.

Multifile applications and applications that access multiple records use the cursor position to determine which box or record an operation applies to. When using IBM 327x and T16-651x terminals, the cursor may be positioned outside of the screen fields recognized by the application. When this happens, the application attempts to locate the record to be affected by the following rules:

1. If the cursor is not positioned in a screen field, the application assumes that it is located in the next screen field (down and to the left) of the display.

2. If the cursor is positioned after the last screen field, the application assumes that it is located in the box and record image where the previous operation took place.

3. If no operations have yet been performed, the operation applies to the first record image in the outermost box.

Standard Screen Format for a Multifile Application

Figure 7-3 shows the record descriptions and ENABLE commands used to generate a sample multifile application.

```
Record Descriptions:

  RECORD parts.
  FILE IS parts   KEY-SEQUENCED.
  02 partnum           PIC 9(4).
  02 partname          PIC X(20).
  02 price             PIC 999999V99.
  KEY 0 IS partnum.
  KEY "pn" IS partname.
  END

  RECORD inventory.
  FILE IS inventry  KEY-SEQUENCED.
  02 inventory-key.
    04 partno            PIC 9(4).
    04 location-num      PIC XXX.
  02 quantity-on-hand  PIC 99999.
  02 reorder-level     PIC X(5).
  KEY 0 IS inventory-key.
  KEY "lo" IS location-num.
  END

ENABLE Commands:

  SET BOX RECORD parts
  ADD BOX parts
  SET BOX RECORD inventory
  ADD BOX inventory
  SET APPL TREE (01 parts
                   02 inventory LINK partnum
                   TO OPTIONAL partno)
  SET APPL PATHCOMFILE multpath
  ADD APPL stock-control
  GENERATE APPL stock-control
```

Figure 7-3. Record Descriptions and ENABLE Commands for a Sample Multifile Application

Figure 7-4 shows the standard screen displayed by this sample application on a T16-651x, T16-652x, or T16-653x terminals.

Standard Screen Format for a Multifile Application

The line numbers that appear to the right of this figure do not appear on the actual screen; these line numbers appear in a table that follows the screen.

```
                                                                  |
                                                                  |
  STOCK-CONTROL                                                (1) |
  Page 1/1                                                     (2) |
  * PARTNUM   ___                                              (3) |
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~                           (4) |
    | * INVENTORY-KEY             |                            (5) |
    |     LOCATION-NUM      ___   |                            (6) |
    |   QUANTITY-ON-HAND  _____   |                            (7) |
    |   REORDER-LEVEL     _____   |                            (8) |
    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~                           (9) |
  + PARTNAME  ____________________                            (10) |
    PRICE     _____.00                                        (11) |
                                                                .  |
                                                                   |
    Ready for Input    F3 for help, Shift F16 to exit          (23) |
                                                               (24) |
```

Line Number	Information Displayed by Application
(1)	The screen title for the application.
(2)	The heading Page 1/1 indicating that this is the first and only screen page.
(3)	The first screen pair for the parent box (parts): * PARTNUM ____ The asterisk (*) that appears to the left of the PARTNUM label identifies this field as a primary key field.

Figure 7-4. Sample Standard Terminal Screen for a Sample Multifile Application (Continued next page)

	Notice that PARTNUM is also the join field of the parts box.
(4)	A series of tilde (~) characters identifies the beginning of a box. The box is aligned with the join field label (PARTNUM) of the parent box.
(5-6)	The screen label and field pairs for a group: * INVENTORY-KEY LOCATION-NUM ___ Note that the application displays only the trailing portion of the inventory-key group. This field is a key for records in the child box that match the join field. The leading portion (partno) does not appear because it is the join field for a child box. Note also the asterisk symbol (*) that appears to the left of the INVENTORY-KEY label. This symbol indicates that the group is a primary key field. A key symbol appears in a child box only if the box contains the trailing portion of a composite key, the leading portion of which is the join field for the box.
(7-8)	The remaining screen pairs for the inventory box
(9)	A series of tilde (~) characters that identify the end of the inventory box
(10-11)	The remaining screen pairs for the parts box
(23)	An informative message

Figure 7-4. Sample Standard Terminal Screen for a Sample Multifile Application (Continued)

With certain exceptions and additions, the rules described previously for the standard screens displayed by single-file applications also apply to the standard screens displayed by multifile applications. These exceptions and additions are:

- The number of boxes that appear on a single screen page affect the number of screen lines available for screen label and field pairs.
- A series of tilde characters (~) identifies the beginning and end of a box.
- A series of vertical line characters (|) indicates the left and right side of a box.
- The join field of a child box does not appear on the screen.
- A child box begins on the screen line following the join field for its parent box. The first label of the child box appears on the next screen line. The application indents at least four screen columns before beginning this screen label.
- If a field will not fit in a child box, the application wraps the field onto the subsequent screen line. The continued field begins in the first column of the box.
- The application might split a box across screen pages.

Standard Screen Formats for IBM-327x Terminals

Screen formats for IBM-327x terminals are the same as those shown for the T16-652x and T16-653x terminals, with the following exception:

- Screen line 23 contains the words Press PF3 for Help, PA2 to exit.
- Screen line 24 displays error message in BRIGHT.

HELP SCREENS

ENABLE applications display two HELP screens. Figures 7-5 and 7-6 show the HELP screens for the T16-651x, T16-652x, and T16-653x. The HELP screen displayed for an IBM-327x terminal substitutes appropriate function key references.

If you generate an application that cannot perform certain operations (delete, insert, read, or update), the HELP screens do not display information about these operations. In addition, if you generate an application that can only display one record in each box, the box operations do not appear on the HELP screens.

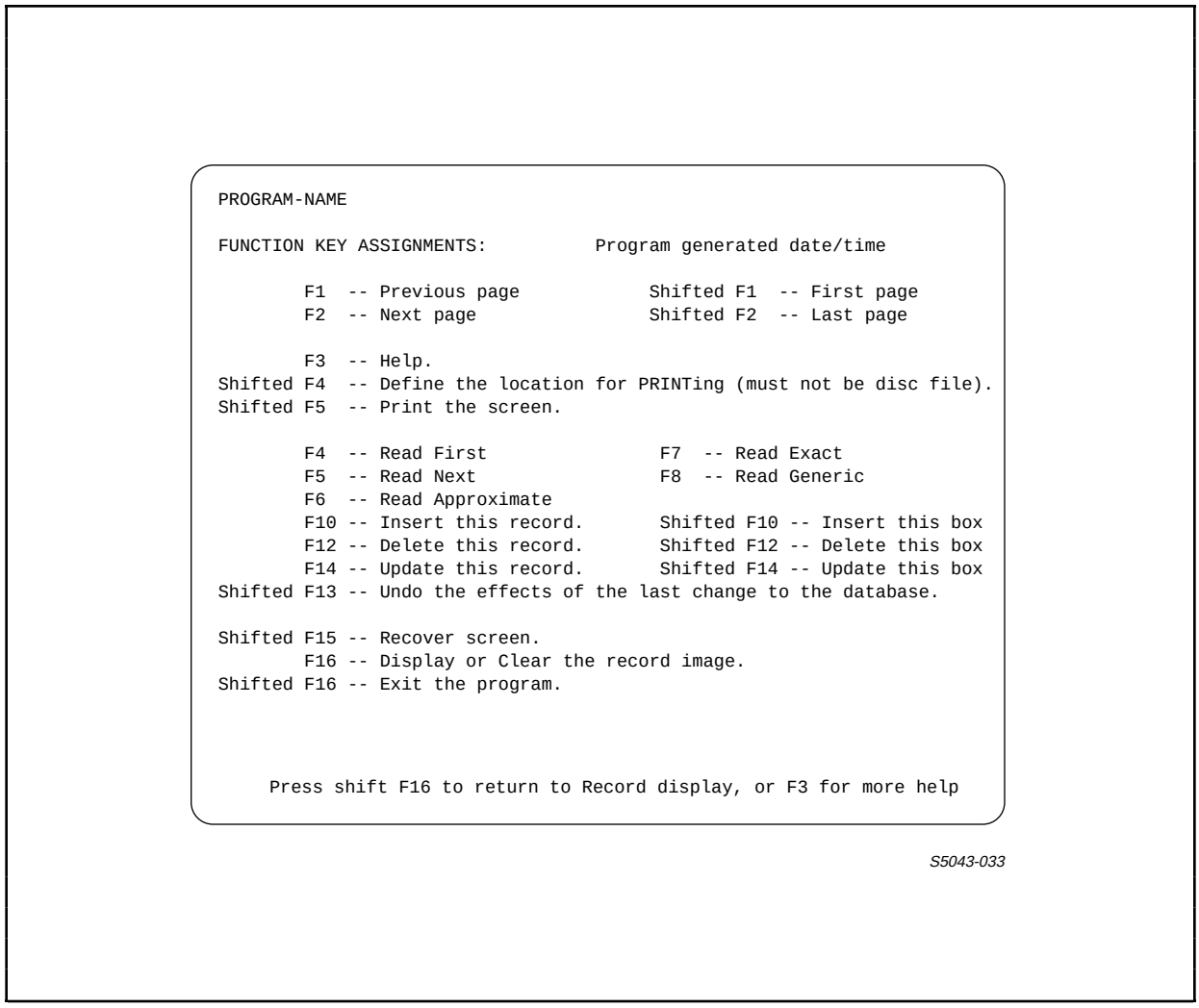


Figure 7-5. First HELP Screen

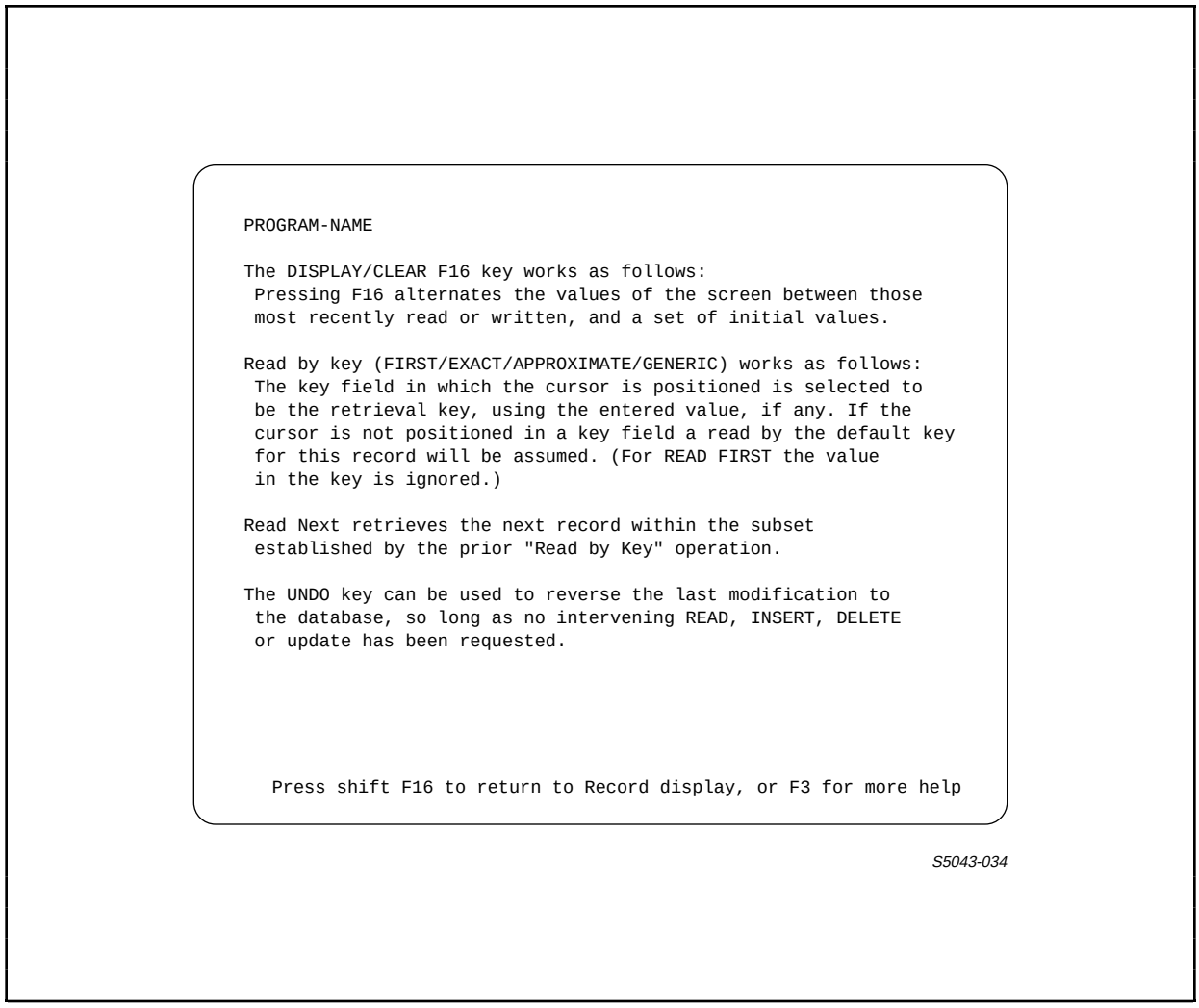


Figure 7-6. Second HELP Screen

SECTION 8

ENABLE FUNCTION KEYS

An ENABLE application displays a screen through which you can read, insert, update or delete records stored in data base files. To request the operation you want, press the appropriate function key on the terminal keyboard. If you use an ENABLE application on a T16-6510, T16-6520, or T16-6530 terminal, you can place an ENABLE template, which identifies the function key operations, across the top of the terminal keyboard. Figure 8-1 illustrates this ENABLE template.

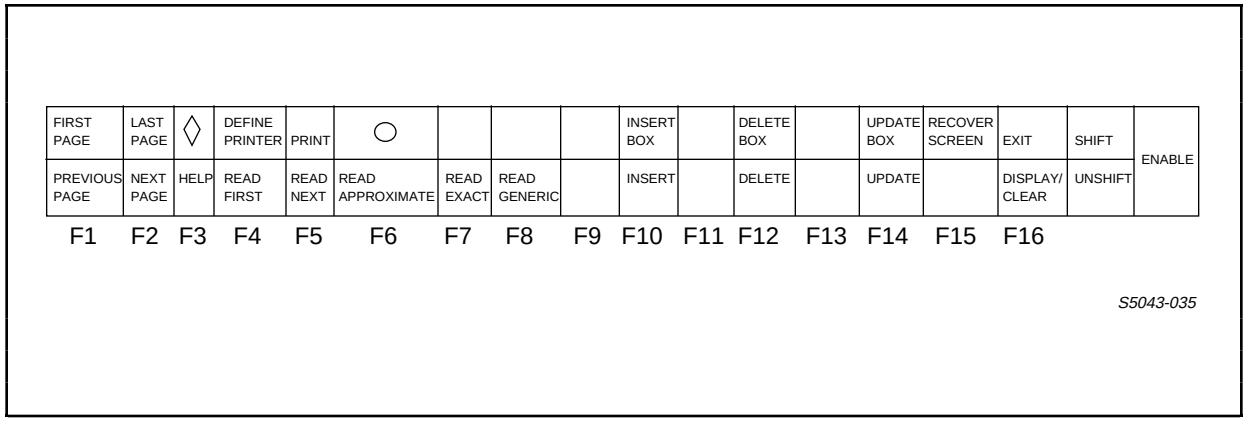


Figure 8-1. The ENABLE Template

Function Key Assignments

This section discusses the ENABLE function keys and the operations assigned to each. Operations on T16-6510, T16-6520, and T16-6530 terminal types are assigned to function keys 1 through 8, 10, 12, 14, and 16, and to shifted function keys 1, 2, 4, 5, 10, and 12 through 16. ENABLE reserves two function keys, SF3 and SF6, for users.

Operations on the IBM-3270 terminal type are assigned to program function keys 1 through 12 and program attention key 2. The ENTER key serves as a shift key for eleven operations corresponding to the shifted F1, F2, F3, F4, F5, F6, F10, F12, F13, F14, and F15 keys.

To use these corresponding shifted functions on an IBM-3270 terminal, do the following:

1. Press the ENTER key first. The screen displays the word SHIFT and the PF keys take on their secondary functions.

2. Press the appropriate PF key. It is not necessary to hold the ENTER key down when the PF key is pressed. You can cancel the effect of the ENTER key by pressing this key a second time.

Table 8-1 briefly describes each function key, its ENABLE template designation, and the operation that you can perform by pressing the key.

Table 8-1. Function Keys (Continued Next Page)

T16-6510/ T16-6520/ T16-6530 Function Key	IBM 3270 PF/PA Key	Template Desig-nation	Operation
F1	PF1	PREVIOUS PAGE	Display the previous screen page or HELP screen.
F1 shifted	ENTER, PF1	FIRST PAGE	Display the first screen page or HELP screen.
F2	PF2	NEXT PAGE	Display the next screen page or HELP screen.
F2 shifted	ENTER, PF2	LAST PAGE	Display the last screen page or HELP screen.
F3	PF3	HELP	Switch between display of record images and HELP screens.
F3 shifted	ENTER, PF3	/\ \/	Reserved for users; usually calls another application.
F4	PF4	READ FIRST	READ FIRST by selected key.

Table 8-1. Function Keys (Continued)

T16-6510/ T16-6520/ T16-6530 Function Key	IBM 3270 PF/PA Key	Template Desig-nation	Operation
F4 shifted	Enter, PF4	DEFINE PRINTER	Define printer.
F5	PF5	READ NEXT	READ NEXT by key selected by previous read operation.
F5 shifted	ENTER, PF5	PRINT	Print display screen.
F6	PF6	READ APPROX	READ APPROXIMATE with indicated key.
F6 shifted	ENTER, PF6	◯	Reserved for users.
F7	PF7	READ EXACT	READ EXACT with indicated key.
F8	PF8	READ GENERIC	READ GENERIC with indicated key.
F10	PF10	INSERT	Insert a record.
F10 shifted	ENTER, PF10	INSERT BOX	Insert record(s) from the current box.

Table 8-1. Function Keys (Continued)

T16-6510/ T16-6520/ T16-6530 Function Key	IBM 3270 PF/PA Key	Template Desig-nation	Operation
F12	PF11	DELETE	Delete a record.
F12 shifted	ENTER, PF11	DELETE BOX	Delete record(s) from the current box.
Fl3 shifted	ENTER, PF7	UNDO	Undo an INSERT, DELETE, or UPDATE.
F14	PF12	UPDATE	Update a record.
F14 shifted	ENTER PF12	UPDATE BOX	Update record(s) from the current box.
F15 shifted	ENTER, PF9	RECOVER SCREEN	Recover display screen.
F16	PF9	DISPLAY/ CLEAR	Display or clear the current record.
F16 shifted	PA2	EXIT	Exit the application.
NEXT PAGE			Display the next screen page or HELP screen.
PREV PAGE			Display the previous screen page or HELP screen.

KEY FIELDS

Many of the operations that you can perform with an ENABLE application involve entering values in key fields. Each record in a key-sequenced file can have:

- A primary key field. The value of a primary key field uniquely identifies a record within a file; for this reason, an ENABLE application will not allow you to enter a value in a primary key field if that value already exists in another record in the file.

- Alternate key fields. The value of an alternate key field identifies records with a certain property. Some alternate key fields require unique values; an ENABLE application will not allow you to enter a duplicate value in such alternate key fields.

Each record in a relative, entry-sequenced, or unstructured file can have:

- A courtesy key field. This key field corresponds to the primary key field of key-sequenced files. A courtesy key field identifies the unique record number of each record within such files. Within these files, the record number has the following significance:

  --For relative files, the record number corresponds to the physical position of a record within the file. The first record position is 0, the next record position is 1, and so forth. A record position exists whether or not a record has been stored in that position.

  --For entry-sequenced files, the record number corresponds to the order in which a record is stored in the file. The first record is 0, the next record is 1, and so forth. Record numbers are always in ascending order, but the numbering sequence is not always in steps of 1. Record 32, for example, could be followed by record 4096. Refer to the ENSCRIBE Programming Manual for more information.

  --For unstructured files, the first record is 0, the next is 1, and so forth. The last record in the file always has the highest record number.

- Alternate key fields. Both relative and entry-sequenced files can have any number of alternate key fields. Alternate key fields are not allowed for unstructured files.

An ENABLE application identifies key fields by placing a special symbol to the left of the label for the field. The application uses an asterisk (*) to identify a primary key field and a plus sign (+) to identify an alternate key field.

FILE OPERATIONS

The following paragraphs describe the operations that you can perform on the data base files accessed by an ENABLE application. These operations are:

READ
INSERT
DELETE
UPDATE

Since slight differences exist in the way that you perform operations with applications that access a single data base file and with applications that access several data base files, the discussion describes operations for these types of applications separately.

Single-File READ Operations

To indicate that the application is to perform a READ operation, you position the cursor in a key field and press the appropriate function key. If you do not position the cursor in a key field, the application selects a default key field for the READ operation. For some READ operations, you must enter a value in the key field.

READ FIRST (F4)

A READ FIRST operation reads the first record (or records) in a file. If you place the cursor in a primary key field, the application reads the first record stored in the file. If you place the cursor in an alternate key field, the application reads the first record associated with the alternate key.

READ NEXT (F5)

A READ NEXT operation reads the next record according to your position within the file. You must have established a position within a file by previously performing a READ operation.

READ APPROX (F6)

A READ APPROX operation reads the first record in the file that has a key value equal to or greater than the value you have entered. If the key field is a composite key (a key field made up of two or more elementary fields), you must enter a value in the first field of the key. For example, suppose that your application displays the following:

```
+ DEPT
    REGNUM     __
    BRANCHNUM  __
```

DEPT is a composite alternate key field. If you want to read a record according to the value of this field, you must enter a value in REGNUM. You can, of course, also enter a value in BRANCHNUM although the application does not require this.

READ EXACT (F7)

A READ EXACT operation reads the first record in a file having a value that matches the value you enter. If you enter a value in a primary key field, the application reads the single record with that value. If you enter a value in an alternate key field, the application reads the first record with that alternate key value.

If the application cannot find a record with the matching key value, it returns a message.

READ GENERIC (F8)

A READ GENERIC operation reads the first record (or records) having a key value that matches the partial value you enter. You cannot perform a READ GENERIC operation on the record number field of a relative, entry-sequenced, or unstructured file. If a key field is defined as USAGE IS COMP or TYPE BINARY in the record description, performing a READ GENERIC operation might return misleading results.

Single-File INSERT Operations

ENABLE applications supply two types of INSERT operations:

- INSERT--To insert a single record
- INSERT BOX--To insert several records at one time (Normally, you only use this type of INSERT operation for applications that display more than one record.)

Before you perform either INSERT operation, you must enter appropriate values in a record (or records).

For INSERT operations on entry-sequenced and unstructured files, the application ignores any value you enter in a record number field. For relative files, you can enter the following in the Record Number field:

- No value to indicate that the record is to be inserted at the first available position
- A positive value to indicate that the record is to be inserted in a particular position
- A negative value to indicate that the record is to be inserted at the end of the file

You can reverse an INSERT operation by requesting an UNDO operation. Refer to the discussion of the UNDO operation later in this section for more information.

INSERT (F10)

An INSERT operation inserts one record in a file. If an application displays more than one record, you must place the cursor within the record to be inserted when you request the INSERT operation.

INSERT BOX (SF10)

An INSERT BOX operation inserts one or more records in a file. When you perform an INSERT BOX operation, you must enter values for at least one record. When the operation completes, the application issues a message stating the number of records that it has inserted. If an error occurs during an INSERT BOX operation, the application:

- Issues a message identifying the problem
- Highlights the fields of the record in error
- Displays a prompt at the top of the screen that asks whether you want to continue processing.

  When this prompt appears you must type either the letter Y or the letter N and press the indicated function key. If you enter the letter Y to continue processing, the application continues the INSERT BOX operation with the next record. If you enter the letter N to stop processing, you can request an UNDO operation and reverse the effects of the INSERT BOX operation.

Single-File DELETE Operations

You must read a record before you can DELETE it. When you request a DELETE operation, the General Server verifies that the indicated record has not been modified since you read it. The General Server locks the record occurrence during this verification. You cannot perform a DELETE operation on an entry-sequenced or unstructured file.

Two types of DELETE operations are available:

- DELETE--To delete a single record

- DELETE BOX--To delete several records at one time (Normally, you use this type of DELETE operation only for applications that display screen pairs for more than one record.)

You can reverse a DELETE operation by requesting an UNDO operation. Refer to the description of the UNDO operation for more information.

DELETE (F12)

A DELETE operation removes a single record from a file. If an application displays more than one record, you must place the cursor within the record to be deleted before you request the DELETE operation.

DELETE BOX (SF12)

A DELETE BOX operation removes one or more records from a file. Before you request a DELETE BOX operation, you must read all of the records you plan to delete. When the application successfully completes the DELETE BOX operation, it issues a message stating the number of records that it deleted. If an error occurs during a DELETE BOX operation, the application:

- Issues a message identifying the problem
- Highlights the fields of the record in error
- Displays a prompt at the top of the screen that asks whether you want to continue processing. When this prompt appears you must type either the letter Y or the letter N and press the indicated function key. If you enter the letter Y to continue processing, the application continues the DELETE BOX operation with the next record. If you enter the letter N to stop processing, you can request an UNDO operation and reverse the effects of the DELETE BOX operation.

Note that a DELETE BOX operation only affects the displayed records. A DELETE BOX operation does not delete other records that have the same join-field value but that are not displayed in the box.

If you restore a record to its initial values (either spaces, zeros, or initial values from the record description), a DELETE BOX operation ignores the record; the application does not try to delete a record with initial values.

Single-File UPDATE Operations

You must read a record before you can request an UPDATE operation. When you request an UPDATE operation, the General Server verifies that the record has not been changed since you read it. The General Server locks the record during this verification.

Two UPDATE operations exist:

- UPDATE--To update a single record
- UPDATE BOX--To update one or more records at the same time (Normally, you use this type of UPDATE operation only for applications that display screen pairs for more than one record.)

You can reverse the effects of an UPDATE operation by requesting an UNDO operation. Refer to the description of the UNDO operation for more information.

UPDATE (F14)

An UPDATE operation updates a single record. If the application displays more than one record, you must place the cursor within the record to be changed before you request the UPDATE operation.

UPDATE BOX (SF14)

An UPDATE BOX operation updates one or more records. When the application successfully completes an UPDATE BOX operation, it displays a message indicating the number of records that it updated. If you request an UPDATE BOX operation but have not changed one or more records within the box, the application does not perform the UPDATE BOX operation on the record or records that remain unchanged.

If an error occurs during an UPDATE BOX operation, the application:

- Issues a message identifying the problem
- Highlights the fields of the record in error
- Displays a prompt at the top of the screen that asks whether you want to continue processing

  When this prompt appears you must type either the letter Y or the letter N and press the indicated function key. If you enter the letter Y to continue processing, the application continues the UPDATE BOX operation with the next record. If you enter the letter N to stop processing, then you can request an UNDO operation and reverse the effects of the UPDATE BOX operation.

Multifile Operations

Multifile applications display screens that contain boxes. Each box displays information from one data base file. The screen itself acts as a box that encloses information from a data base file.

Figure 8-2 shows a sample screen displayed by a multifile application. Note that the boxes appear to be nested. Boxes are of graduated sizes, each box fitting within a larger one. The outermost box is bounded by the edge of the terminal screen. Nested boxes can contain other boxes. A box that contains another box is called a containing box. The box that fits within the immediately larger box is called a nested box. A box can be both a containing box and a nested box.

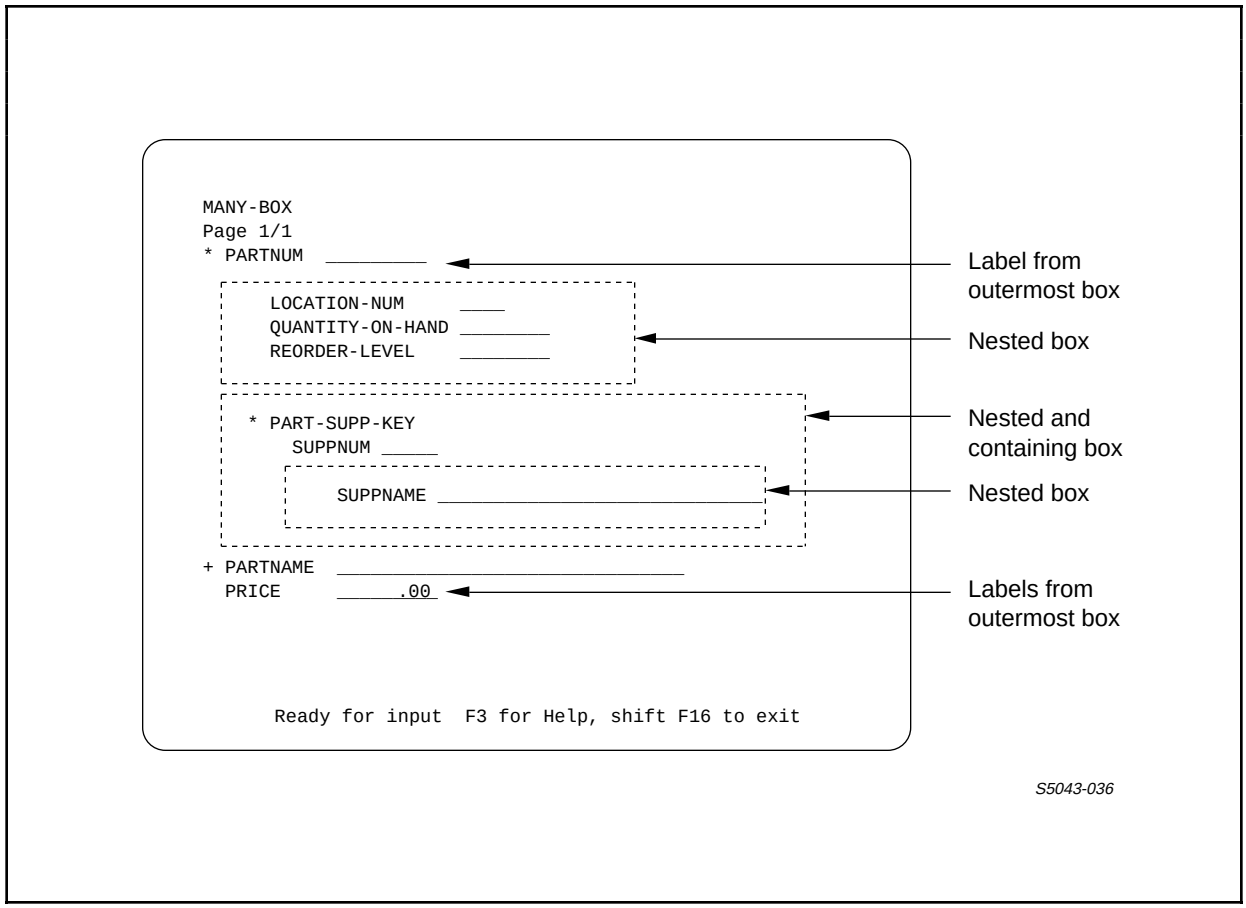


Figure 8-2. Sample Terminal Screen Displayed by a Multifile Application

If a screen contains several boxes, you can request operations for one box at a time. To identify the box upon which the operation is to be performed, position the cursor within the box.

You request operations for these boxes in much the same manner as you request these operations when an application displays a single box. In most cases, you can request DELETE, INSERT, READ, and UPDATE operations as described earlier in this section. The following paragraphs provide special information that applies only to applications that display two or more boxes.

Multifile READ Operations

You can read a record for an outermost box by requesting any READ operation (READ APPROX, READ EXACT, READ FIRST, READ GENERIC, or READ NEXT) described earlier in this section.

If you want to read a record for a nested box, you must first request either a READ or INSERT operation for its containing box. When you READ or INSERT a record for the containing box, you provide a join-field value that the application uses when you read a record for the nested box. (Refer to the description of the TREE attribute in Section 4 for the definition of a join field.)

If you add a nested box with the FILL attribute set to ON, the application automatically performs a READ FIRST operation on a nested box whenever you request a READ, UPDATE, or INSERT operation for its containing box. If the application automatically performs a READ FIRST operation on a nested box, you can continue reading records for that box by requesting READ NEXT operations.

If a key field appears within a nested box, you can read records for the nested box by entering a value in this field and requesting any READ operation.

If a key field does not appear within a nested box, you can read records for that box only by placing the cursor within the box and requesting a READ FIRST or READ NEXT operation.

When you follow a READ operation on a nested box with any operation (READ, UPDATE, DELETE, or INSERT) that changes the join-field value of the containing box, the information in the nested box:

- Disappears if the nested box was added with the FILL attribute set to OFF
- Is replaced with any existing new information if the nested box was added with the FILL attribute set to ON

Multifile INSERT Operations

When an application displays several boxes, you must request an insert operation for the outermost box before you can request an insert operation for any other box.

You must also insert a record in a containing box before you can insert any records in nested boxes.

Multifile DELETE Operations

When you request a DELETE operation, delete the record (or records) from a nested box before you delete any records from its containing box.

If you delete a record from a containing box before you delete any records from its nested boxes, the application cannot access the records for its nested boxes because the record in the containing box with a matching join-field value no longer exists. Since you must read a record before you can delete it, you cannot delete any records for the nested box.

Multifile UPDATE Operations

With one exception, you can request UPDATE operations in the same manner that you request these operations for applications that display records from a single file. The exception is when you want to update the join-field value of a containing box; to do this:

1. Delete all of the matching records from its nested box (or boxes). To be sure that you have deleted all the records with matching join-field values, request READ NEXT operations until the application indicates that no more records can be found.

2. Use any UPDATE operation to change the record in the containing box.

3. Use any INSERT operation to replace the records in the nested box (or boxes).

Undo (SF13) Operations

You can undo a file operation (DELETE, DELETE BOX, INSERT, INSERT BOX, UPDATE, or UPDATE BOX) by requesting an UNDO operation.

To undo a DELETE opertion, the application must insert the deleted record. The application can undo a DELETE operation if another file operation has not intervened between the DELETE operation and the UNDO request.

To undo an UPDATE operation, the application must replace the old record. The application can undo an UPDATE operation if:

- Another file operation has not intervened between the UPDATE operation and the UNDO request
- The updated record has not been modified by another application

To undo an INSERT operation, the application must delete the inserted record. The application can undo an INSERT operation if:

- Another file operation has not intervened between the INSERT operation and the UNDO request
- The inserted record has not been modified by another application
- The INSERT operation was not performed on an entry-sequenced or unstructured file. (The application must delete the record to undo the INSERT operation, but ENSCRIBE, the data base record manager supplied by Tandem, does not allow DELETE operations on entry-sequenced or unstructured files.)

Print Operations

You can print the current contents of the screen by requesting two operations:

- DEFINE PRINTER
- PRINT

DEFINE PRINTER (SF4)

A DEFINE PRINTER operation defines the terminal, printer, or process that is to receive the current screen image. When you request a DEFINE PRINTER operation, the application displays a prompt on line 2 of the screen. You must then enter the name of the terminal, printer, or process and press SF4 again.

PRINT (SF5)

A PRINT operation prints the current screen image on the terminal, printer, or process defined with a previous DEFINE PRINTER request. The application displays a message while performing the PRINT operation.

RECOVER SCREEN (SF15) Operations

When the application performs a RECOVER SCREEN operation, it redisplays data items stored in the working storage area of the application. These data items might not necessarily correspond to values that were on the screen at the time that you requested the RECOVER SCREEN operation.

Special Operations (SF3 and SF6)

ENABLE reserves two function keys (SF3 and SF6) for users. To activate these function keys, you must modify the SCREEN COBOL source code after application generation.

If the SCREEN COBOL source code has not been modified to make use of these keys, the application returns a message stating that the keys are not supported when you press either SF3 or SF6.

If these keys are active, their purpose is application dependent.

APPENDIX A

SYNTAX SUMMARY

This appendix summarizes the syntax of ENABLE commands, attributes and operating commands. A page reference accompanies each item.

```
RESET [ APPL ] { [ <attribute> ]       }                3-21
      [ BOX  ] { [ ABILITY     ]       }
               { [ FORMAT      ]       }
               { [ INTEGRITY   ]       }
               { [ OTHER       ] , ... }
               { *                     }


SET [ APPL ] <attribute> <value>                        3-26
    [ BOX  ]

  [ , <attribute> <value> ] ...

  or

SET [ APPL ] LIKE <object> [ , <attribute> <value> ] ...
    [ BOX  ]


SHOW [ APPL ] [ <attribute> ]                           3-31
     [ BOX  ] [ ABIlITY     ]
              [ FORMAT      ]
              [ INTEGRITY   ]
              [ OTHER       ]
              [ *           ]
```

Attributes:

```
BOXTITLE { 1 } <string-literal>                          4-8
         { 2 }
         { 3 }


CHECKDATA { ON  }                                        4-11
          { OFF }


DATAFILE <data-file-name>                                4-13


DELETE { ON  }                                           4-15
       { OFF }


DICTIONARY { <subvolume>                        }        4-17
           { $<volume>.<subvolume>              }
           { \<system>.$<volume>.<subvolume> }
```

```
EXCLUDE { <qualified-field-name>                 }        4-19
        { ( <qualified-field-name> [ , ... ] ) }

FILL { ON  }                                              4-24
     { OFF }

FLAG { <flag-number> } <flag-value>                       4-28
     { *             }

HEADINGS { DDLFIELDNAMES  }                               4-30
         { DDLHEADINGS    }
         { NULL           }

INCLUDE {  <qualified-field-name>                 }       4-32
        {  ( <qualified-field-name> [ , ... ] ) }

INSERT { ON  }                                            4-36
       { OFF }

NONSTOP { ON  }                                           4-37
        { OFF }

PATHCOMFILE <file-name> [!]                               4-39

PATHCOMSKELETON <skeleton-file-name>                      4-41

READ { ON  }                                              4-43
     { OFF }

RECORD <external-record-name>                             4-45

SCOBOLCOMPILER [ <compiler-name> ]                        4-47

SCOBOLLIST [ <file-name> [!] ]                            4-49

SCOBOLOBJECT [ <file-name> ]                              4-51
```

Operating Commands:

```
CMDSYS [ \<system-name> ]                                          5-5


CMDVOL { $<volume>              }                                  5-6
       { $<volume>.<subvolume> }
       { <subvolume>            }


ENV [ CMDSYS  ]                                                    5-8
    [ CMDVOL  ]
    [ OBEYSYS ]
    [ OBEYVOL ]
    [ SYSTEM  ]
    [ VOLUME  ]


EXIT                                                               5-9


FC                                                                5-10

  R<replacement-string>

  I<insertion-string>

  D


HELP [ <command-name>      ]                                      5-13
     [ {<}<symbol-name>{>} ]


OBEY <filename>                                                   5-14


OBEYSYS [ \<system-name> ]                                        5-16


OBEYVOL { $<volume>              }                                5-17
        { $<volume>.<subvolume> }
        { <subvolume>            }
```

```
OUT <file-name>                                                    5-19

  or

<command> /OUT <file-name> / <parameter>


SYSTEM [ \<system-name> ]                                          5-21


VOLUME { $<volume>              }                                  5-22
       { $<volume>.<subvolume> }
       { <subvolume>            }
```

APPENDIX B

ENABLE MESSAGES

This appendix lists error and warning messages that may be issued in response to ENABLE commands during application generation, or during execution of an ENABLE application.

Unless specifically noted as a warning, all messages are error messages. Error messages signify that an error in processing has occurred. A warning message signifies that a questionable condition exists. These conditions are handled as follows:

ERROR ENABLE prefixes error messages with the label "*** ERROR ***."

An ERROR is fatal to the operation being attempted; in the case of a GENERATE command, ENABLE does not create a PATHCOM command file, SCREEN COBOL source code, or SCREEN COBOL object code.

If ENABLE is running in interactive mode, it issues a prompt. If running in noninteractive mode, it terminates.

WARNING ENABLE prefixes warning messages with the label "*** WARNING ***."

A warning indicates a questionable condition that does not halt processing of the requested application.

ENABLE calls the SCREEN COBOL compiler internally; therefore, you could receive a message from the SCREEN COBOL compiler that is not listed in this appendix. Refer to the SCREEN COBOL Reference Manual for a list of SCREEN COBOL messages.

Messages may also be received from the GUARDIAN operating system. Refer to the GUARDIAN Operating System Programmer's Guide for information about these messages.

Table B-1 lists the messages that ENABLE might issue in response to commands or during application generation.

Table B-1. ENABLE Error and Warning Messages
(Continued next page)

Message	Meaning
A flag in the range 0 to 99 must be specified	The value supplied for the <flag-num> parameter of the FLAG attribute is invalid.
All fields have been excluded from this box	No fields exist in the box. You specified all fields in the record description when you supplied a value for the EXCLUDE attribute. Change the value of the EXCLUDE attribute.
All key fields have been excluded from this box	No primary or alternate key field exists in a box that represents a key-sequenced file. Either you supplied the EXCLUDE attribute with a value that eliminated all key fields, or you did not specify a key field when you supplied a value for the INCLUDE attribute. If the file is entry-sequenced, relative, or unstructured, you might have excluded both the courtesy key and all alternate keys. If you supplied a value for INCLUDE, you must explicitly include the courtesy key. Use the SHOW command to check the value of both the INCLUDE and EXCLUDE attributes.

Table B-1. ENABLE Error and Warning Messages (Continued)

Message	Meaning
An item size exceeds the maximum supported	Either a PIC or a TYPE clause in the record description identifies a field that is more than 256 bytes long.
An unsupported data type appears in the record - field: <field-name>	The record description shows the named field with an invalid data type (BINARY and CHARACTER are valid data types), or the named field is a numeric item with more than 18 characters.
At least one file operation option must be selected for this BOX	The value of all file ABILITY attributes (FILL, DELETE, INSERT, READ, and UPDATE) are OFF for the box. The value of at least one of these attributes must be ON.
Box contains field of the same name	A box name must not be the same as any field within the file that the box represents. Use a different name for the box.
Boxtitle is too long to fit in BOX	The string literal that you supplied as a value for the BOXTITLE attribute does not fit within the box on the terminal screen.
Cannot specify SCOBOLLIST with compilation suppressed	You supplied a value for SCOBOLLIST, but the value of SCOBOLOBJECT indicates no SCREEN COBOL compilation. Change the value of one of these attributes.

Table B-1. ENABLE Error and Warning Messages (Continued)

Message	Meaning
Could not obtain extended memory (ALLOCATESEGMENT) error: <err-num>	The GUARDIAN operating system could not obtain the number of extended memory pages requested for the object and attribute tables. Use the EXTPAGES parameter to reduce the number of extended memory pages allocated. Try allocating 100 or 200 pages.
Data file name was not specified	The record description does not identify a file name, and you did not supply a value for the DATAFILE attribute. Supply a value for this attribute.
Data item is too long to fit in box: <field-name>	The named field contains too many characters to fit within the box on the terminal screen.
The DDL record description exceeds 2046 bytes	The record description indicates that the record contains more characters than allowed for an ENABLE application.
Delete has been set off for this box because of filetype	Warning message. ENABLE has set DELETE to OFF because this box represents a file that is either entry-sequenced or unstructured. Delete operations are not allowed for files of these file types.

Table B-1. ENABLE Error and Warning Messages (Continued)

Message	Meaning
Dictionary file number mmmm File name : <file-name> File management error code = <err-num>	The indicated I/O error occurred on the named dictionary file.
Dictionary is out of date: please convert	The value of the DICTIONARY attribute identifies a dictionary that was created by a version of DDL earlier than D00. To use this dictionary, recompile its source code with a newer version of DDL.
Duplicate parameter	The same attribute has been named more than once in a SET command, a RESET command, an ADD command, or a GENERATE command.
Effective input line is too long	The maximum length of 528 bytes has been exceeded; there are too many continuation lines.
ENABLE internal error	A system software error occurred. Notify your system analyst.
ENABLE internal file error File management error code = <err-num>	A file management system or sequential I/O procedure error occurred.
ENABLE tables overflow allocated extended memory	Both the attribute table and the object table have overflowed.
Error in communicating with ENABLE server File management error code = <err-num>	A file management system or sequential I/O procedure error has occurred.

Table B-1. ENABLE Error and Warning Messages (Continued)

Message	Meaning
Exclude and Include cannot both be selected	ENABLE does not allow you to supply a value for both the EXCLUDE and INCLUDE attributes. Reset the value of one of these attributes.
** FAILURE 18 ** DICTIONARY OVERFLOW *** ERROR *** SCOBOL COMPILATION ERRORS ...	The SCREEN COBOL compiler did not have enough data space to compile the generated program.
File error while accessing SCOBOL compiler process File management error code = <err-num>	A file management system or sequential I/O procedure error occurred.
Field corresponding to a unique key has been excluded	For a box with INSERT ON, you excluded a unique alternate key or the primary key of a key-sequenced file by doing one of the following: • Specifying the key(s) or a portion of the key as a value for the EXCLUDE attribute. • Not specifying the key(s) or a portion of the key as a value for the INCLUDE attribute.
Flags must be set to a value between 0 and 255	You supplied an invalid value for the <flag-value> paramater of the FLAG attribute. Supply a valid value for this parameter.

Table B-1. ENABLE Error and Warning Messages (Continued)

Message	Meaning
Garbled PATHCOM skeleton -- Edit line nbr = aaaa.bbb	The PATHCOM skeleton file does not conform to its expected structure. If you did not modify the PATHCOM skeleton file, notify your system analyst.
Garbled SCOBOL skeleton -- Edit line nbr = aaaa.bbb	The SCREEN COBOL skeleton file does not conform to its expected structure. If you not modify the SCREEN COBOL skeleton file; notify your system analyst.
Identifier too long	You supplied a name that exceeds 30 characters.
Illegal OBEY file	The obey file does not have a valid name or cannot be accessed.
Illegal OUT file - ignored	The OUT file name is invalid; the listing is not redirected.
Initial value too long - discarded initial value for: <field-name>	Warning message. The record description defines the named field with an initial value that has more than 30 characters. ENABLE does not use the initial value.
INSERT is not allowed with this linked field and filetype	The INSERT attribute must be OFF in a child box that represents either an entry-sequenced or unstructured file when the join field of the child box is the courtesy key (the record number).

Table B-1. ENABLE Error and Warning Messages (Continued)

Message	Meaning
Invalid APPL parameter	ENABLE does not recognize the specified application attribute. Check your spelling of this attribute.
Invalid BOX command	The object type of a GENERATE command must not be BOX. Either include the keyword APPL with the GENERATE command or use the ASSUME command to make APPL the default object type.
Invalid BOX parameter	ENABLE does not recognize the specified box attribute. If the attribute is an application attribute, the current object type must be APPL. If the attribute is a box attribute, check your spelling of this attribute.
Invalid boxtitle number	You specified an invalid number with one of the BOXTITLE attributes. Valid numbers are 1, 2, or 3.
Invalid file name	A file name you specified does not conform to system file-naming standards.
Invalid level number	An invalid value appears as a level number for the value of the TREE attribute. Valid values for level numbers range from 1 to 50.

Table B-1. ENABLE Error and Warning Messages (Continued)

Message	Meaning
Invalid name - PATHCOM reserved word	You used a PATHCOM reserved word as a name. PATHCOM reserved words are illegal in certain constructs when a PATHCOM command file is being generated.
Invalid name - reserved ENABLE prefix T9155-	A name begins with the ENABLE prefix T9155-.
Invalid name - SCOBOL reserved word	A name is a SCREEN COBOL reserved word.
Invalid Subvolume name	The name is not a valid subvolume name or is not valid for the current system name.
Invalid syntax	The sequence of input characters does not conform to ENABLE language syntax. A ^ symbol indicates the element an error.
Invalid System name	The name is not a known system same.
INVOKE returned error code nnnn	ENABLE could not access the dictionary because of the indicated error. Record the error and notify your system analyst.
Level numbers are improperly sequenced in TREE	The level numbers for a tree structure command are incorrectly sequenced. Check for a level number that is lower numerically than the level number of the first box identified as a value for the TREE attribute.

Table B-1. ENABLE Error and Warning Messages (Continued)

Message	Meaning
Link field appears in an OCCURS item: <field-name>	The named join field of a box is: • Modified by an OCCURS clause in the record description • Part of a group modified with an OCCURS clause in the record description You cannot specify an OCCURS item as a join field.
Link field data lengths are incompatible: <field-name>	The join field of a child box is shorter than the join field of the parent box.
Link field data types are incompatible: <field-name>	The LINK option of the TREE attribute specifies join fields with incompatible data types.
Link must be optional	The keyword OPTIONAL was omitted from a LINK option of the TREE attribute.
Linked field does not appear in box: <field-name>	The named join field does not exist in the child box. Check the spelling of the join field name. If the name is spelled correctly, use the INFO BOX command to check the value of either the INCLUDE or EXCLUDE attribute. You might have excluded the join field by supplying it as a value for the EXCLUDE attribute or by not supplying it as a value for the INCLUDE attribute.

Table B-1. ENABLE Error and Warning Messages (Continued)

Message	Meaning
Linked field is not a key field: <field-name>	The join field of a child box must be a primary key field, an alternate key field, a courtesy key field, or the leading (leftmost) portion of a composite key.
Linked field must not be reordered or incomplete: <field-name>	If a join field is a group field, the elementary items of that group must not be reordered. Use the INFO BOX command to check the value of the INCLUDE and EXCLUDE commands for the child box. If the join field is a group that contains FILLER items, these items are automatically excluded by ENABLE. Groups containing FILLER items should not be used as join fields, although the leftmost field (if not a FILLER item) may be.
Linking field does not appear in box: <field-name>	The named join field does not exist in the parent box. First, check the spelling of the join-field name. If the name is spelled correctly, use the INFO BOX command to check the value of the INCLUDE or EXCLUDE attribute for the box. You must not exclude a join field either explicitly (by supplying it as a value for the EXCLUDE attribute) or implicitly (by supplying it as a value for the INCLUDE attribute).

Table B-1. ENABLE Error and Warning Messages (Continued)

Message	Meaning
Linking field must not be reordered or incomplete: <field-name>	If a join field is a group field, the elementary items of that group must not be reordered. Use the INFO BOX command to check the value of the INCLUDE or EXCLUDE attributes for the parent box. If the join field is a group that contains FILLER items, these items are automatically excluded by ENABLE. Groups containing FILLER items should not be used as join fields, although the leftmost field (if not a FILLER item) may be.
List file error File management error code = <err-num>	A file management system or sequential I/O procedure error occurred. Notify your system manager.
List file name error	The name of the list file does not conform to system file-naming standards.
Mismatched attributes in shared SERVERCLASS: <serverclass-name>	The same server class of the General Server cannot be shared by boxes when: • Some boxes have TMF ON and other boxes have TMF OFF • Some boxes have NONSTOP ON and other boxes have NONSTOP OFF • Some boxes have NONSTOP ON and other boxes have TMF ON

Table B-1. ENABLE Error and Warning Messages (Continued)

Message	Meaning
	Supply a value for the SERVERCLASS attribute to identify a different copy of the General Server for boxes with different integrity attribute values.
Must have READ with UPDATE or DELETE	The UPDATE or DELETE option is ON, but READ is OFF. You must supply ON as a value of the READ attribute if either UPDATE or DELETE is ON.
Name not found: <field-name>	ENABLE cannot find the named field in the record description.
Name not sufficiently qualified to avoid ambiguity: <field-name>	The named field requires qualification. Refer to the discussion of ENABLE command conventions in the reference manual.
No field is left to be displayed in box: <box-name>	The value of the EXCLUDE attribute indicates that all fields are to be excluded from the named box or that only the join field is left in the box. at least one field must appear in a box on the terminal screen.
No microcode for ENABLE in this CPU	ENABLE microcode has not been installed in the CPU.
NONSTOP and TMF cannot both be selected	Both NONSTOP and TMF are ON. reset the value of one of these attributes.

Table B-1. ENABLE Error and Warning Messages (Continued)

Message	Meaning
No SCOBOL object code was generated	Warning message. ENABLE did not generate object code because you omitted the <file-name> parameter when you supplied a value for the SCOBOLOBJECT attribute.
No SCOBOL source or object files were generated	Warning message. ENABLE did not generate either SCREEN COBOL source code or SCREEN COBOL object code because of the value of the SCOBOLSOURCE and SCOBOLOBJECT attributes.
Number too large, too small, or not an integer	You entered a number (a level number, size, or flag) that is invalid.
OBEY nesting exceeds maximum	Obey-file nesting exceeds four levels.
OCCURS nesting too deep	In a record description, the OCCURS clause nesting exceeds 4 levels.
OCCURS value is too big	In a record description, a field is described with an OCCURS clause that indicates more than 999 occurrences.
Object in use	If you are trying to add an object, an object with this name already exists in the object table. Each object (application or box) must have a unique name. If you are trying to delete a box, the box is being used by an application.

Table B-1. ENABLE Error and Warning Messages (Continued)

Message	Meaning
PATHCOM file already exists	The value of the PATHCOMFILE attribute identifies an existing file, but you did not include the exclamation point symbol (!) to force an overwrite.
PATHCOM file error File management error code = nnnn	The indicated file management system or sequential I/O procedure error occurred.
PATHCOM Program name truncated to ...	Warning message. ENABLE has truncated the PATHCOM program name to the indicated 15 characters.
PATHCOM skeleton file error File management error code = <err-num>	The indicated file management or sequential I/O procedure error has occurred on the PATHCOM skeleton file.
PATHCOM skeleton file name error	The file name set for the PATHCOMSKELETON attribute does not conform to system file-naming standards.
Program cannot be generated with box size specified	ENABLE cannot generate an application with the value set for the SIZE attribute. The value could be too small, too large, or not an integer.
RECORD name must be specified	The value of the RECORD attribute is null. You must supply a value for this attribute.

Table B-1. ENABLE Error and Warning Messages (Continued)

Message	Meaning
Same box already used in TREE: <box-name>	The named box appears more than once as the value of the TREE attribute.
SCOBOL compilation errors -- see file <file-name>	The SCREEN COBOL compiler could not compile the SCREEN COBOL source code. The indicated file contains the listing generated by the SCREEN COBOL compiler.
SCOBOL object file name error	The value of the SCOBOLOBJECT attribute identifies a file name that does not conform to system file naming standards.
SCOBOL object file name must be < 6 char	The value of the SCOBOLOBJECT attribute identifies a file name that exceeds five characters.
SCOBOL process ABENDed -- source and listing on files <file-name-1>, <file-name-2>	The SCREEN COBOL compiler process terminated abnormally. The generated source code and SCREEN COBOL listing (if any) are on the indicated files.
SCOBOL skeleton file error File management error code = <err-num>	The indicated file management system or sequential I/O procedure error occurred on the SCREEN COBOL skeleton.
SCOBOL skeleton file name error	The file name supplied for the SCOBOLSKELETON attribute does not conform to system file naming standards.

Table B-1. ENABLE Error and Warning Messages (Continued)

Message	Meaning
SCOBOL source file already exists	You supplied the name of an existing file as the value of the SCOBOLSOURCE attribute, but you did not include the exclamation point symbol (!) to force an overwrite. Either change the file name or include the exclamation point symbol.
SCOBOL source file error File management error code = <err-num>	The indicated file management system or sequential I/O procedure error occurred on the SCREEN COBOL source file.
SERVER name must be < 16 char	The value of the SERVERCLASS attribute is a name that exceeds 15 characters. Change the value of this attribute.
Specified APPL not found: <appl-name>	The named application has not been entered in the object table.
Specified BOX not found: <box-name>	The named box has not been entered in the object table.
Specified Record not found: <record-description-name>	ENABLE cannot find the named record description in the dictionary.
System unknown or not available	Either the system does not exist, or ENABLE cannot access it.

Table B-1. ENABLE Error and Warning Messages (Continued)

Message	Meaning
The generated PATHCOM file must be edited.	Warning message. You must edit the PATHCOM file before using it to configure a PATHWAY system.
The maximum box size for this record is <num>	The indicated number is the maximum value to which the SIZE attribute can be set for this record.
The same record element has been referenced twice: <field-name>	The named field has been supplied twice as a value for either the INCLUDE or EXCLUDE attribute. This error also appears if you enter a group name and an element within the group as a value for either attribute.
This version of ENABLE cannot be run on a TNS system.	This version of ENABLE must be run on either a TNS II or a TXP system.
Title too long to fit on screen	The string literal supplied for the TITLE attribute exceeds 79 characters in length.
Tree statement references undeclared BOX	The TREE statement contains the name of a box that does not exist in the object table. Use the INFO command to be sure you added the box to the object table. Check the spelling of the box name.

Table B-1. ENABLE Error and Warning Messages (Continued)

Message	Meaning
Unable to access dictionary File name : <file-name> File management error code = <err-num>	ENABLE either could not find the dictionary files or could not access them. <err-num> identifies the cause of the error.
Unable to access file. File name: <file-name> File management error code = <err-num>	ENABLE cannot access the file because of either a file-management system error, or a sequential I/O procedure error. <err-num> identifies the cause of the error.
Unterminated continuation line	ENABLE encountered an end-of-file condition when a continuation line was expected.
Unterminated string	The string literal supplied for either the TITLE attribute, or a BOXTITLE attribute is not terminated by a quotation mark.
Very low on extended memory; please DELETE unwanted objects	An overflow condition will occur for the ENABLE tables unless you use the DELETE command to delete any unnecessary boxes and applications.
Wrong version of PATHCOM skeleton	The PATHCOM skeleton file does not match the current ENABLE product version.

Table B-1. ENABLE Error and Warning Messages (Continued)

Message	Meaning
Wrong version of SCOBOL skeleton	The SCREEN COBOL skeleton file does not match the current ENABLE product version.

Table B-2 lists messages that an ENABLE application might issue while running.

Table B-2. Application Run-Time Error Messages
(Continued next page)

Message	Meaning
Alternate key is gone.	You tried to read a file using an alternate key and the General Server cannot find, open, or read the alternate key file.
An invalid printer was specified.	You specified a device that was not a printer, terminal, or process, in response to the DEFINE PRINTER prompt.
Default record is not acceptable	You tried to insert a record entirely composed of default values, or you tried to update a record so that it is entirely composed of default values. Such records are not allowed.
DELETE failed. File error code = nnnn	A GUARDIAN-ENSCRIBE file management error occurred. Record the error and see your data administrator.
DELETE failed. Record is gone	You tried to delete a record that no longer exists; the record might have been deleted by some other application since you last read it.
DELETE failed. Record is locked	You tried to delete a record that is locked by some other process. Try the operation again.

Table B-2. Application Run-Time Error Messages (Continued)

Message	Meaning
Fatal error occurred during printout	During I/O to the print device, a GUARDIAN file error code indicating a fatal error condition was returned. Reenter the name of the printer and try the operation again. If the operation fails again, see your data administrator.
File OPEN error. AUDIT/TMF param mismatch	Either the General Server is being run with TMF OFF and the file is audited by TMF, or the General Server is being run with TMF ON and the file is not audited by TMF. Record the error and see your data administrator.
File OPEN error. Data file security violation (048)	A security violation occurred at file-open time.
File OPEN error. Data file was in use (012)	The file could not be accessed at open time because another program was using it.
File OPEN error. Data file was not found (011)	The file could not be found at open time.
File OPEN error. ENABLE version mismatch	The program is calling a General Server module that is from the wrong version of ENABLE.

Table B-2. Application Run-Time Error Messages (Continued)

Message	Meaning
File OPEN error. File name was not assigned.	No ASSIGN naming the logical record referred to was supplied to the called server class. Check the PATHCOM command file for a SET SERVER ASSIGN command.
File OPEN error. File system error code = <err-num>	A file management error occurred on open.
File OPEN error. General Server needs more memory.	The General Server needs more memory to open the files. Start the General Server with MEM 64, reduce the number of files assigned to the General Server, or assign some of the files to another serverclass of the General Server and regenerate the program.
File OPEN error. NONSTOP and TMF were both selected	The General Server is being run with both NONSTOP and TMF ON.
File OPEN error. Regenerate program: file has changed	The organization of the data base file does not agree with the organization described in the record description used to generate the ENABLE application. Any of the following might have changed: the record length, the file type, or the offset and length of the key fields. Either the file or the program must be corrected so that the same record description is used for both the application and the data file.

Table B-2. Application Run-Time Error Messages (Continued)

Message	Meaning
Files must not be changed prior to DELETE.	You tried a delete operation after changing some value in a field of the record.
INSERT failed. Duplicate key	A key-field value of the record you tried to insert duplicated an existing key-field value in the data base. Primary keys can never be duplicated; alternate keys can be defined to accept or disallow duplicate values.
INSERT failed. File error code = nnnn	A GUARDIAN-ENSCRIBE file-management error occurred. Record the error and see your data administrator.

Table B-2. Application Run-Time Error Messages (Continued)

Message	Meaning
INSERT OK, but DELETE on old image failed.	You tried to update a record and change the primary key. When you change the primary key with an update operation, the General Server must insert a new record with the new primary key and delete the old record with the old primary key. The General Server was able to insert the new record but the old record was not available and therefore could not be deleted. If the old record cannot be deleted, the General Server tries to delete the new record. If you receive this message, the General Server did not delete either record. Record the error and see your data administrator. If this situation is not corrected, the data base will be in an inconsistent state.
Invalid KEY SPECIFIER	An invalid key ID was entered. The key was not one of the known keys.
INVALID NUMBER FORMAT	You entered characters in a numeric field, entered a digit in the sign position of a signed field, or omitted a required decimal. This error is posted by PATHWAY.

Table B-2. Application Run-Time Error Messages (Continued)

Message	Meaning
Invalid numeric field(s) displayed as zero: data is corrupt	If you have just read a record and this message appears, the highlighted fields contain invalid data that the application displays as zeros. Record the error and see your data administrator.
Invalid RECORD length	Modification of either the SCREEN COBOL skeleton or the SCREEN COBOL source code has resulted in the entry of an invalid record length. The length was not within the limits imposed by the file structure.
Join field value was not acceptable.	You entered an invalid value (such as a -1) in the join field of the containing box when the nested box represents a relative file. The join field of the containing box is the last field on the screen before the child box.
Join field was changed on the screen but not updated.	You entered a new join-field value on the screen for the containing box but did not request an update operation for this box. You cannot read a record for the nested box until either you read or insert a record in the containing box or you return the join field value to the value read from the data base.

Table B-2. Application Run-Time Error Messages (Continued)

Message	Meaning
No base screen was displayed	Modifications to either the SCREEN COBOL skeleton or the SCREEN COBOL source code resulted in the omission of a required DISPLAY BASE statement. The current screen is undefined. Correct the error and recompile the SCREEN COBOL source code.
No changes were specified.	You tried an update operation without first changing the record read from the data base.
No detail screens can be accessed from this box.	You cannot call another application to obtain detailed information for this box because the application does not support a call for this box.
No key field was identified.	You tried a Read Next operation before reading the first record. Precede a Read Next operation by one of the following operations: Read First, Read Approximate, Read Exact, or Read Generic.
No parent item.	You tried to perform an operation on a nested box without first performing a read or insert operation on the containing box.
Nothing to delete.	You tried to delete a record without previously reading the record from the data base.

Table B-2. Application Run-Time Error Messages (Continued)

Message	Meaning
Nothing to update.	You tried to update a record without previously reading the record from the data base.
OPEN error: <file-name> File system error code = <err-code>	The specified error occurred on open for the named file. Record the error and see your data administrator.
Printer requires attention.	The printer is not ready.
Program error: Corrupt data file info	The request to the General Server contains incorrect information about a data file that was previously opened successfully by the same requester.
Record is locked	The record you tried to read has been locked by another application.
Record not found.	The record you tried to read does not exist in the data base.
Screen recovery. Some entries to this screen may have been lost	You either pressed the SCREEN RECOVER key or a terminal I/O was detected and recovery was successful. Values you entered on the current screen before the failure might have to be re-entered.
Selected function key is not supported by this application.	You requested an operation that is not supported by this application, or you pressed a function key that is not assigned to an operation.

Table B-2. Application Run-Time Error Messages (Continued)

Message	Meaning
SEND Error status value = nnn	A PATHWAY error has occurred. Record the error and see your data administrator for interpretation of the error code associated with the PATHWAY SEND verb.
This operation is not supported	You attempted an operation that is not supported by this application.
UNDO cannot be executed at this time	You tried to undo an operation either after requesting a read operation, or when there was no operation to be undone.
Unknown FUNCTION-CODE	Modifications to either the SCREEN COBOL skeleton or the SCREEN COBOL source code have resulted in a FUNCTION-CODE that is unknown to the General Server.
Unknown TRANS-CODE	Modifications to either the SCREEN COBOL skeleton or the SCREEN COBOL source code have resulted in a TRANS-CODE that is unknown to the General Server.
Update conflict. Record has been reread	You tried to update a record that has been modified by some other application since you last read it. The record has been reread by ENABLE, and the new value appears on the screen.
UPDATE failed. Duplicate key	You entered a duplicate value for a key that is declared to be unique.

Table B-2. Application Run-Time Error Messages (Continued)

Message	Meaning
UPDATE failed. File error code = nnnn	A GUARDIAN-ENSCRIBE file management error occurred. Record the error and see your data administrator.
UPDATE failed. Primary key must not be changed.	You tried to update a record in an entry-sequenced, relative, or unstructured file and changed the record number key field. To change the Record Number field for a record in a relative file, you must delete the old record and insert a new one. You cannot alter the value of the record number field for an unstructured or entry-sequenced file.
UPDATE failed. Record is gone	You tried to update a record that no longer exists.
UPDATE failed. Record is locked	You tried to update a record that is locked by some other process. Try the operation again.

APPENDIX C

RESERVED WORDS

This appendix lists the reserved words in three categories: ENABLE reserved words, SCREEN COBOL reserved words, and PATHWAY reserved words.

Figure C-1 lists the ENABLE reserved words. You cannot use these words as either box or application names.

ENABLE Reserved Words		
APPL	BOX	KEY

Figure C-1. ENABLE Reserved Words

SCREEN COBOL Reserved Words

Figure C-2 lists the SCREEN COBOL reserved words. You cannot use these words as application names.

SCREEN COBOL Reserved Words

ABORT
ABORT-INPUT
ABORT-TRANSACTION
ABSENT
ACCEPT
ACCESS
ADD
ADVANCING
ADVISORY
AFTER
ALARM
ALL
ALPHABETIC
ALSO
ALTER
ALTERNATE
AND
APPROXIMATE
ARE
AREA
AREAS
ASCENDING
ASSIGN
AT
ATTR
AUDIBLE
AUTHOR

BASE
BE
BEFORE
BEGIN-TRANSACTION
BLANK
BLOCK
BOTTOM
BY

CALL
CANCEL
CD
CF
CH
CHARACTER
CHARACTERS
CHARACTER-SET
CHECKPOINT
CLEAR
CLOCK-UNITS
CLOSE
COBOL
CODE
CODE-SET
COLLATING
COLUMN
COLUMNS
COMMA
COMMUNICATION
COMP
COMPUTATIONAL
COMPUTE
CONFIGURATION
CONTAINS
CONTROL
CONTROLS
CONVERSATIONAL
CONVERSION
COPY
CORR
CORRESPONDING
COUNT
CROSSREF
CURRE
CURRENCY
DATA
DATE
DATE-COMPILED
DATE-WRITTEN
DAY
DE
DEBUG-CONTENTS
DEBUG-ITEM
DEBUG-LINE
DEBUG-NAME
DEBUG-SUB-1
DEBUG-SUB-2
DEBUG-SUB-3
DUBUGGING
DECIMAL-POINT
DECLARATIVES
DELAY
DELETE
DELIMITED
DELIMITER
DEPENDING
DESCENDING
DESTINATION
DETAIL
DIAGNOSTIC-ALLOWED
DISABLE
DISPLAY
DIVIDE
DIVISION
DOWN
DUPLICATES
DYNAMIC

EGI
ELSE
EMI
ENABLE
END
END-OF-INPUT
END-OF-PAGE
END-TRANSACTION
ENTER
ENVIRONMENT
EOP
EQUAL
ERROR
ERROR-ENHANCEMENT
ESCAPE
ESI
EVERY
EXCEPTION
EXCLUSIVE
EXIT
EXTEND

FD
FIELD-SEPARATOR
FILE
FILE-CONTROL
FILL
FILLER
FINAL
FIRST
FIXED-LENGTH
FOOTING

Figure C-2. SCREEN COBOL Reserved Words (Continued next page)

FOR
FROM
FULL

GENERATE
GENERIC
GIVING
GO
GREATER
GROUP
GROUP-SEPARATOR

HEADING
HIGH-VALUE
HIGH-VALUES
I-O
I-O-CONTROL
I-O-ERROR
IDENTIFICATION
IF
IN
INDEX
INDEXED
INDICATE
INITIAL
INITIATE
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INTO
INVALID
IS

JUST
JUSTIFIED

KEY

LABEL
LAST
LEADING
LEFT
LENGTH
LESS
LIKE
LIMIT
LIMITS
LINAGE
LINAGE-COUNTER
LINE
LINE-COUNTER
LINES
LINKABE
LOCK
LOCKFILE
LOGICAL-TERMINAL-NAME
LOW-VALUE
LOW-VALUES

MEMORY
MERGE
MESSAGE
MODE
MODEM
MODULES
MOVE
MULTIPLE
MULTIPLY
MUST

NATIVE
NEGATIVE
NEW-CURSOR
NEW-CURSOR-COL
NEW-CURSOR-ROW
NEXT
NO
NOSHADOW
NOT
NUMBER
NUMERIC
NUMERIC-SHIFT

OBJECT-COMPUTER
OCCURS
OF
OFF
OFFSET
OLD-CURSOR
OLD-CURSOR-COL
OLD-CURSOR-ROW
OMITTED
ON
ONE
OPEN
OPTIONAL
OR
ORGANIZATION
OUTPUT
OVERFLOW
OVERLAY
PAGE
PAGE-COUNTER
PATHWAY
PERFORM
PF
PH
PIC
PICTURE
PLUS
POINTER
POSITION
POSITIVE
PRINT
PRINTING
PROCEDURE
PROCEDURES
PROCEED
PROGRAM
PROGRAM-ID
PROGRAM-STATUS
PROGRAM-STATUS-1
PROGRAM-STATUS-2
PROMPT
PROTECT

QUEUE
QUOTE
QUOTES

RANDOM
RD
READ
RECEIVE
RECEIVE-CONTROL
RECONNECT
RECORD
RECORDS
RECOVERY
REDEFINES
REDISPLAY
REEL
REFERENCES
RELATIVE
RELEASE
REMAINDER
REMOVAL
RENAMES
REPLACING
REPLY
REPORT
REPORTING

Figure C-2. SCREEN COBOL Reserved Words (Continued)

REPORTS
RERUN
RESERVE
RESET
RESTART-COUNTER
RESTART-INPUT
RESTART-TRANSACTION
RETURN
REVERSED
REWIND
REWRITE
RF
RH
RIGHT
ROUNDED
RUN

SAME
SCREEN
SCREEN-CONTROL
SCROLL
SD
SEARCH
SECTION
SECURITY
SEGMENT
SEGMENT-LIMIT
SELECT
SEND
SENTENCE
SEPARATE
SEQUENCE
SEQUENTIAL
SET
SHADOWED
SHARED
SIGN
SIZE
SKIP
SKIPPING
SORT
SORT-MERGE
SOURCE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
SPACES
STANDARD
STANDARD-1
START
STARTBACKUP
STATUS
STOP
STOP-MODE
STRING
SUB-QUEUE-1
SUB-QUEUE-2
SUB-QUEUE-3
SUBTRACT
SUM
SUPPRESS
SYMBOLIC
SYNC
SYNCDEPTH
SYNCHRONIZED
SYSTEM

TAB
TABLE
TAL
TALLYING
TAPE
TELL-ALLOWED
TEMP
TEMPORARY
TERMINAL
TERMINAL-FILENAME
TERMINAL-PRINTER
TERMINATE
TERMINATION-STATUS
TERMINATION-SUBSTATUS
TEXT
THAN
THROUGH
THRU
TIME
TIMEOUT
TIMES
TO
TOP
TRAILING
TRANSACTION-ID
TRANSPARENT
TURN

UNDER
UNIT
UNLOCK
UNLOCKFILE
UNLOCKRECORD
UNSTRING
UNTIL
UP
UPON
UPSHIFT
USAGE
USE
USER
USING

VALUE
VALUES
VARYING

WHEN
WITH
WORDS
WORKING-STORAGE
WRITE

YIELDS

ZERO
ZEROES

S5043-037

Figure C-2. SCREEN COBOL Reserved Words (Continued)

Figure C-3 lists the PATHWAY reserved words. You cannot use these words as application names.

PATHWAY Reserved Words

ABORT	INTERVAL	PRI
ADD	IOPROTOCOL	PRINTER
ALTER	IS-ATTACHED	PROCESS
ASSIGN	LIKE	PROCESSES
ASSOCIATIVE	LOG1	PROGRAM
	LOG2	PROTECTED
BACKUPCPU		
BLOCK	MAXASSIGNS	REC
	MAXEXTERNALTCPS	REFRESH-CODE
CHECK-DIRECTORY	MAXLINKS	RESET
CMDVOL	MAXPARAMS	RESUME
CODE	MAXPATHCOMS	
COLD	MAXPATHWAYS	SECS
CONVERSATIONAL	MAXPROGRAMS	SECURITY
COOL	MAXREPLY	SERVER
COUNT	MAXSERVERCLASSES	SERVERPOOL
CPUS	MAXSERVERPROCESSES	SET
CREATEDELAY	MAXSERVERS	SHARED
	MAXSTARTUPS	SHOW
DUBUG	MAXTCPS	SHUTDOWN
DELETE	MAXTELLQUEUE	START
DELETEDELAY	MAXTELLS	STARTUP
DETAIL	MAXTERMDATA	STATS
	MAXTERMS	STATUS
ERROR-ABORT	MAXTMFRESTARTS	STOP
EXCLUSIVE	MINS	SUSPEND
EXIT		SWAP
EXT'	NONSTOP	SWITCH
	NUMSTATIC	
FC		TCLPROG
FILE	O	TCP
FREEZE	OBEY	TELL
	OBEYVOL	TERM
HELP	OFF	TERMBUF
HOMETERM	ON	TERMPOOL
HRS	OPEN	THAW
	OUT	TMF
I-O	OUTPUT	TYPE
IN	OWNER	
INFO		VOLUME
INITIAL	PARAM	
INPUT	PATHMON	WAIT
INSPECT	PATHWAY	WARM

S5043-038

Figure C-3. PATHWAY Reserved Words

APPENDIX D

MODIFYING THE SCREEN COBOL SKELETON FILE

You can modify a copy of the SCREEN COBOL skeleton file provided by ENABLE and use this modified version to generate an application. The SCREEN COBOL skeleton file contains source text for a SCREEN COBOL program that is complete except for the data items defined by DDL record descriptions and options included by the ENABLE compiler.

Special command lines in the skeleton drive the ENABLE processing. These command lines are interspersed with text lines. Before you attempt to modify SCREEN COBOL code that is controlled by these special command lines, you should understand the effect of the modification. When you modify a special command line or add a new command line, you do so at your own risk. Because the skeleton contains comments at the beginning that explain how these special command lines work, this manual does not explain them.

If you make an error when you modify a command line, ENABLE terminates with a diagnostic message indicating the edit line in the skeleton file containing the error.

To modify the skeleton, you must obtain a copy of the skeleton file. The skeleton file resides on a file named ENABAPPS. Normally, this file resides on $SYSTEM.SYSTEM. If ENABAPPS does not reside on this volume and subvolume, ask your system manager to tell you the location of this file.

Duplicate a copy of ENABAPPS and make your modifications to this copy. To use the copy when you generate an application, use the SET APPL SCOBOLSKELETON command.

The following list describes some possible modifications to the SCREEN COBOL skeleton file:

1. You can replace literals in the text. For example, you can modify the skeleton so that the application will display messages in a language other than English. If the modified literal is longer than the original, you might need to modify the affected SCREEN COBOL data items.

2. You can remove the T9155- prefix from the variables in the skeleton provided no name conflict will result. Removing this prefix does improve readability of the SCREEN COBOL source code; it is present in the skeleton solely to avoid conflict with the PROGRAM-ID name and field names from the DDL record description.

3. You can associate user flags with a portion of code. In a command line, you can use the tests EQUAL, LESS THAN, and GREATER THAN to test the value of a user flag. Depending on the result of the test, ENABLE either uses, or does not use the associated code. You could, for example, modify the SCREEN SECTION of the skeleton to include the UPSHIFT attribute for alphanumeric fields. If you associate this portion of the code with a user flag, you can turn UPSHIFT off and on with the SET FLAG command. For example, suppose that you include the following in your ENABLE commands used to generate an application:

   ```
   SET BOX FLAG 10 1
   ```

   If the skeleton contains the following:

   ```
   % 1 2   94  110  2 <
         UPSHIFT INPUT-OUTPUT
   % 1 0   94
   ```

   the code UPSHIFT INPUT-OUTPUT will be included in the specified block of code for the generated application. Note that the SET FLAG command identifies flag number 10. This flag number corresponds to the 110 in the first command line. (In the skeleton, user-flag numbers are from 100 to 199.) The line of code is included because the SET FLAG command sets this flag to 1 and the command line indicates that the code should be included if user flag 10 is less than (<) 2. ENABLE includes the code between the two command lines.

4. Commands in the application skeleton determine the generation of the SCREEN SECTION. By modifying the application skeleton, the screen attributes generated into programs can be controlled. For example, you could set all key fields to BLINK, change the symbols (* and + by default) that identify primary and alternate keys, or define new characters to be used as fill characters.

If you make modifications to the skeleton that are suitable for all users at your installation, you can replace ENABAPPS with the modified version, which thus becomes the default skeleton.

CAUTIONS

If you replace the default skeleton with a modified version, you do so AT YOUR OWN RISK.

Any skeletons modified for use with previous versions of ENABLE cannot be used with the present version of ENABLE.

Future versions of ENABLE may include further enhancements to the SCREEN COBOL skeleton; these may not be compatible with the skeleton provided with the present version of ENABLE.

APPENDIX E

CHARACTERISTICS OF THE GENERAL SERVER

The General Server is a server that can be called from any requester requiring access to a data base file. You can use the General Server:

- In a PATHWAY system with a requester program generated by ENABLE
- In a PATHWAY system with a requester program that you have written
- Outside of a PATHWAY system with a requester program that you have written

Using the General Server with a Requestor Generated by ENABLE

When you use the General Server with a requester generated by ENABLE, you normally use a PATHCOM command file also generated by ENABLE to establish your PATHWAY system. The PATHCOM command file contains all of the commands necessary to start the General Server and provide it with the information it needs to open and access data base files.

Optionally, you can edit the PATHCOM command file generated by ENABLE and insert the following commands:

SET SERVER (PARAM ALLFILES ON)	Indicates that the General Server must open all assigned files before processing any requests. If the General Server cannot open an assigned file, it terminates.
SET SERVER (PARAM WHOLEFILES ON)	Indicates that the General Server is to issue an error message if it cannot open an alternate key file or a partition of a data file.

When a requester generated with ENABLE opens the General Server, the General Server compares the record description passed from the requester with the file characteristics of the file being opened. This includes the record length, the file type, and the offset and length of the key fields known to the requester. If the record description information and the file characteristics differ, the General Server issues an error message to the requester. The requester then reports this error message to the user.

If you want to override this comparison process, alter the generated SCREEN COBOL source code to move blanks into T9155-VERSION before the the OPEN transaction and recompile.

Using the General Server with a User-Written Requester

If you use the General Server with a requester that you have written, the requester must meet the requirements of the General Server. These requirements are described later in this appendix.

STARTING THE GENERAL SERVER

The General Server resides on an object file named ENABLEGS. Normally, this file resides on $SYSTEM.SYSTEM. If ENABLEGS does not reside on this volume and subvolume, ask your system manager for the location of the file.

If you plan to use the General Server within a PATHWAY system, you must include the following in the series of SERVER commands that describe the General Server:

```
SET SERVER PROGRAM $SYSTEM.SYSTEM.ENABLEGS
```

To start the General Server outside of a PATHWAY environment, you use the command interpreter RUN command. Refer to the GUARDIAN Operating System User's Guide information about this command.

Providing File Assignments

The General Server can open up to 32 data base files. You must provide ENABLEGS with the assign specifications that it needs to open these files. If you want to use the General Server within a PATHWAY system, these assign specifications have the following form:

```
SET SERVER (ASSIGN <logical-name>, <file-name>
  [ , <file-attributes> ] ...)
```

Refer to the PATHWAY System Management Reference Manual for more information about this command

If you use the General Server outside of a PATHWAY environment, you must provide the assign specifications before you start the General Server. You can provide these specifications by using the command interpreter ASSIGN command. Refer to the GUARDIAN Operating System User's Guide for information about this command.

In either case, you must supply logical file names in the assign specification that match the logical file names in the requester program. These are the logical file names that are passed in an OPEN transaction.

When you specify a physical file name in the assign specification, you can use a fully expanded file name or rely on the defaults to expand the file name. If the physical file is an unstructured file, you must supply the record size as a parameter in the assign specification. If the physical file has any other file type, ENABLEGS ignores the record size.

At initialization, ENABLEGS opens the data base files with the access mode indicated in the ASSIGN specification. If you do not specify an access mode in an assign specification, ENABLEGS opens the file for input-output with shared access. The assigned files must exist, be available, and allow access with the access mode you specify. The assigned files must also allow access by the process-accessor ID under which ENABLEGS is running as follows:

- When running under PATHWAY, the ID is the process-accessor ID of the PATHMON process that controls this server class. By default, PATHMON has the process-accessor ID of the person who started it.

- When not running under PATHWAY, the ID is the user ID of the user who started the ENABLEGS process.

Providing Parameters for the General Server

The General Server recognizes the following parameters:

- ALLFILES [ON]
  [OFF]

  ON indicates that ENABLEGS must open all assigned files before processing any requests. If ENABLEGS cannot open one of the assigned files, it terminates.

  OFF indicates that ENABLEGS opens any assigned files that it can before processing a request. If ENABLEGS cannot open a particular file, it cannot process a request for that file.

  The default for this parameter is OFF.

- NONSTOP [ON]
  [OFF]

  ON indicates that the General Server is to run as a NonStop process pair. If you supply ON as a value for this parameter and the primary server fails, the new primary will wait for the next request before attempting to start a backup. If a backup cannot be started, the primary will run unprotected until such time as a backup can successfully be started in the specified backup CPU.

  OFF indicates that the General Server does not run as a NonStop process pair.

  The default for this parameter is OFF.

NOTE

Do not set the NONSTOP parameter to ON if the General Server is to process TMF audited files. Any attempt to open a server started with both TMF and NONSTOP parameters ON will result in a FILE-ERROR-CODE of 305.

- TMF [ON]
  [OFF]

ON indicates that the data base files to be accessed by ENABLEGS are audited by TMF. A data base file audited by TMF is flagged for special handling in preparation for recovery efforts. For use under PATHWAY, all the files accessed by a given server class must either be audited by TMF or be non-audited files.

OFF indicates that none of the data base files to be accessed by ENABLEGS is audited by TMF.

The default for this parameter OFF.

- WHOLEFILES [ON]
  [OFF]

  ON indicates that ENABLEGS is to issue an error message if it cannot access an alternate key file or a partition of a data file.

  OFF indicates that ENABLEGS does not issue an error message if it cannot access an alternate key file or a partition of a data file.

  The default for this parameter is OFF.

If you use the General Server in a PATHWAY system, you provide parameters by using the SET SERVER PARAM command. Refer to the PATHWAY System Management Reference Manual for information about this command.

To pass parameters to the General Server outside of a PATHWAY system, you must use the command interpreter PARAM command. Refer to the GUARDIAN Operating System User's Guide for more information about this command.

THE GENERAL SERVER AND LOCKING

The General Server does not perform file-level locking. The General Server locks a record immediately before issuing an UPDATE or DELETE and verifies that the old record image presented by the caller matches the locked record from the file before proceeding with the modification. If the General Server detects a mismatch, it returns the current value of the record with an error status.

GENERAL SERVER REQUIREMENTS

You can use the General Server with requester programs that you have written. These programs must conform to the external specifications detailed in the following paragraphs.

Opening the General Server

You can open the General Server with a nowait depth of 0 or 1 and a sync depth of 0 or 1. The sync depth mechanism of the General Server emulates that of the disc process in that only replies to nonretryable requests are saved for retransmission. UPDATE, DELETE, and INSERT are nonretryable operations; OPEN and READ operations get reexecuted, regardless of sync ID. If a request for a nonretryable operation arrives with a sync ID older than the one saved by the General Server for that requester, the General Server responds with GUARDIAN file error 39.

General Server Request Format

General Server requests consist of a fixed prefix common to all types of transactions and a variable portion dependent upon the transaction code (TRANS-CODE) entry. The request prefix consists of a 24-byte PATHWAY header, a 54-byte General Server header, and a 96-byte context buffer. Figure E-1 shows a DDL structure that describes the format of the General Server request prefix.

Table E-1 lists the valid values for the TRANS-CODE field and the types of transactions to which these values apply.

```
DEF PW-HEADER.
02   REPLY-CODE            PIC S9(4) COMP.
02   APPLICATION-CODE      PIC XX.        (not used)
02   FUNCTION-CODE         PIC XX.
02   TRANS-CODE            PIC 99.
02   TERM-ID               PIC X(15).     (not used)
02   LOG-REQUEST           PIC X.         (not used)
     END

DEF GS-HEADER.

02   ERROR-STATUS.
   03  ERROR-CODE          PIC S9(4) COMP.
   03  ERROR-SUB-CODE      PIC S9(4) COMP.
   03  FILE-ERROR-CODE     PIC S9(4) COMP.
02   FILE-NAME             PIC X(30).
02   FILE-NUMBER           PIC S9(4) COMP.
02   FILE-POSITION.
   03  RECORD-KEY          PIC S9(9) COMP.
   03  PRIOR-POSITION      PIC X(4).
02   KEY-SPECIFIER         PIC XX.
02   COMPARE-LENGTH        PIC S9(4) COMP.
02   RECORD-LENGTH         PIC S9(4) COMP.
02   RECORD-AREA-LENGTH    PIC S9(4) COMP.
     END

DEF GS-CONTEXT.
02   CONTEXT-BUFFER        PICTURE X(96).
02   CONTEXT REDEFINES CONTEXT-BUFFER.
  03   VERSION             PIC X(4).
  03   LOGICAL-KEY-LENGTH  PIC 9(4) COMP.
  03   FILLER              PIC X(90).
       END
```

Figure E-1. General Server Request Structure in DDL-Compatible Format

Table E-1. Transaction Codes

Code Value	Transaction
0	OPEN
1	READ
2	UPDATE
3	DELETE
4	INSERT

The first transaction that you request from the General Server must be an OPEN to obtain FILE-NUMBER.

There is no close-file transaction. When all calling requester programs have issued a close against the General Server process (that is, have terminated), the General Server closes the files and terminates. As long as any requester has the General Server open, the General Server process continues executing and maintains all data files open.

Table E-2 lists the fields in the General Server request structure that apply to specific transactions.

Table E-2. Fields and Transaction Codes

Field Name	Transaction Codes
FUNCTION-CODE	1,4
FILE-NAME	all codes
FILE-NUMBER	all codes
FILE-POSITION	1,2,3
KEY-SPECIFIER	1,2,3
COMPARE-LENGTH	1
RECORD-LENGTH	0,2,3,4
RECORD-AREA-LENGTH	0,1,2,3
LOGICAL-KEY-LENGTH	1

ENABLEGS ignores fields that are not required. For example, ENABLEGS will ignore COMPARE-LENGTH except on a random READ (TRANS-CODE =1).

Usage-Dependent Fields

As Figure E-2 shows, nine fields in the prefix are associated with transaction codes. These fields are:

- FUNCTION-CODE
- FILE-NAME
- FILE-NUMBER
- FILE-POSITION
- KEY-SPECIFIER
- COMPARE-LENGTH
- RECORD-LENGTH
- RECORD-AREA-LENGTH
- LOGICAL-KEY-LENGTH

Specific information about each field is given in the following paragraphs.

General Server header fields that are not specifically required for a transaction should have the values as returned by the previous transaction.

FUNCTION-CODE Field

The requester must supply a value for this field only for a READ operation (code 1) and INSERT operation (code 4).

For READ operations, the value supplied can be any of the following:

FR	READ FIRST (by selected key from KEY-SPECIFIER)
EX	READ EXACT (by selected key from KEY-SPECIFIER)
AP	READ APPROXIMATE (by selected key from KEY-SPECIFIER)
GN	READ GENERIC (with a portion of the key)
AG	READ APPROXIMATE GENERIC (a READ APPROXIMATE with the value of the full key, while establishing a new key set using a portion of that key)
NX	READ NEXT (within the current key set, or within the file)
NN	READ NEXT (with a new key set, but starting with a READ APPROXIMATE on the record supplied)

For INSERT operations, the value supplied can be any of the following:

blank	INSERT in the first available space
EX	INSERT at the indicated record key position (relative files only)
LA	INSERT at the end of the file (relative files only)

FILE-NAME Field

This field applies to all transaction codes. For an open operation, the requester must supply the 30-character logical name of the data file to be opened. The requester need not change this value for subsequent operations.

FILE-NUMBER Field

This field applies to all transaction codes. ENABLEGS returns the identifying file number in this field after a successful file-open transaction. File numbers are 0 through 31. The requester should not change this value.

FILE-POSITION Field

This group applies to a READ operation (code 1), UPDATE operation (code 2), and DELETE operation (code 3). The group contains the file position value returned from the previous call to ENABLEGS with the following exceptions:

READ EXACT by record key on relative files and entry-sequenced files	The requester must supply the value of the relative record number in the RECORD-KEY field of FILE-POSITION.
READ EXACT by relative byte address on unstructured files	The requester must supply the relative byte address in the RECORD-KEY field of FILE-POSITION.

The General Server returns the following to PRIOR-POSITION:

- The first two bytes contain the positioning mode for the key specifier.
- The second two bytes contain the compare length.

KEY-SPECIFIER Field

This field applies to a READ operation (code 1), UPDATE operation (code 2), and DELETE operation (code 3). The requester must supply the two-byte key specifier for the current key of reference in this field. Supplying binary zero indicates the primary key.

COMPARE-LENGTH Field

This field applies to a random READ operation (code 1). A random READ operation is one with a FUNCTION-CODE equal to EX, AP, AG, or GN. For each random READ, the requester must set this field to the byte length of the leading portion of the supplied key value. For READ EXACT and READ APPROXIMATE only, a zero value indicates the total defined key length is to be used. For READ APPROXIMATE Generic, the value of this field corresponds to the length of the key to be used to establish a keyset for future READ operations.

RECORD-LENGTH Field

This field applies to an open operation (code 0), a read operation (code 1), an UPDATE operation (code 2), a DELETE operation (code 3), and an INSERT operation (code 4).

If the requester sets this field to zero before the open operation, the General Server returns the expected value for RECORD-LENGTH in this field. The expected value is the byte length of the record for the file being opened. For unstructured files without the ODDUNSTR attribute, this value is rounded up by one byte if the byte length is an odd number.

The requester should not change the value of this field for any other operation. The General Server sets this field to the length of the record read in for a READ operation.

An open operation sets this field to the expected value of RECORD-AREA-LENGTH. If the requester sets this field to zero subsequent to the open operation, the General Server uses and returns the expected value. The expected value for this field is the value of RECORD-LENGTH if RECORD-LENGTH is an even number or RECORD-LENGTH + 1 if RECORD-LENGTH is an odd number. This area could exceed the length of the record by one byte because RECORD-IMAGE-2, the new record image, must always begin on a word boundary.

LOGICAL-KEY-LENGTH Field

This code applies to a READ operation (code 1) with a FUNCTION-CODE equal to AG. For each READ, the requester must set this field to the byte length of the portion of the supplied key value to be used for the READ APPROXIMATE. COMPARE-LENGTH defines the generic portion of the full key that is used to establish a new key set.

Context Buffer

The General Server uses the VERSION field of CONTEXT to carry version information. If the requester supplies a version number in the form of B00, for example, the General Server supplies the capabilities of version B00. The requester must supply this information with each request.

The buffer is also included for future extension of the interface to servers that need to retain additional context information.

Variable Portion of the General Server Request

The variable portion of a General Server request consists of zero, one, or two record images, depending on the transaction code value in TRANS-CODE. Figure E-2 shows the variable portion of a General Server request.

```
RECORD-IMAGE     PIC X(n)  (codes 1,2,3,4)

RECORD-IMAGE-2   PIC X(m)  (code 2)

 The presence of record images is dependent on specific
 transactions; relevant transaction codes are shown in
 parentheses.  RECORD-IMAGE-2, for example, is present only
 on UPDATE (TRANS-CODE=2).

 You do not need to supply RECORD-IMAGE for code 0.  You do
 not need to supply RECORD-IMAGE for code 1 if the operation
 is a READ FIRST or if the data file is not key-sequenced.
 You must supply RECORD-IMAGE for all other operations.

 RECORD-IMAGEs always begin on a word boundary.
```

Figure E-2. Variable Portion of a General Server Request

The following list describes what the requester must supply in RECORD-IMAGE and RECORD-IMAGE-2 (see Figure E-2) depending upon the transaction.

- READ SEQUENTIAL (TRANS-CODE=1 and FUNCTION-CODE=NX or NN):

  --The requester must supply the current record image in RECORD-IMAGE.

  --For READ NEXT by a new key (FUNCTION-CODE=NN), the requester must indicate the selected key in KEY-SPECIFIER.

- READ RANDOM (TRANS-CODE=1 and FUNCTION-CODE=FR, EX, AP, AG, or GN):

  --For symbolic keys (alternate keys and the primary key for key-sequenced files), the requester must supply the candidate key value in its appropriate position in RECORD-IMAGE; the General Server ignores the remainder of the record. The requester must set KEY-SPECIFIER to indicate the appropriate key. If FUNCTION-CODE=FR, however, the requester need not supply a record image.

  --The requester must set RECORD-KEY to the primary key value for files that are not key-sequenced but are being read by the primary key. The requester must set KEY-SPECIFIER to binary zero to indicate the primary key.

- UPDATE (TRANS-CODE=2):

  --The requester must supply the old record image in RECORD-IMAGE.

  --The requester must supply the new record image in RECORD-IMAGE-2. The new record follows the old record but always begins on a word boundary.

  --The requester must supply the byte length of the old record image in RECORD-AREA-LENGTH. This might not be the length of the record itself if padding of one byte is necessary to position RECORD-IMAGE-2 on a word boundary.

NOTE

When the primary key is changed by an UPDATE operation, the General Server actually performs an insert of the new image followed by a DELETE of the old image. If the DELETE of the old image fails because of error, both the new and old images are left in the data base. If the data base file is audited by TMF, the requester can execute an ABORT-TRANSACTION to restore the data base to its former state. If the data base file is not audited by TMF, you must take other action to correct the situation. If the situation is not corrected, the data base could be left in an inconsistent state.

- INSERT (TRANS-CODE=4):

  The requester supplies the new record image in RECORD-IMAGE.

- DELETE (TRANS-CODE=3):

  The requester supplies the old record image in RECORD-IMAGE.

SEND Processing

Table E-3 shows the fields and corresponding values that must be supplied for SEND processing. The table is divided according to the operation requested.

Table E-3. Values Expected by the General Server for SEND Processing (Continued Next Page)

Operation	Field or Record	Value
OPEN	TRANS-CODE	00
	FILE-NAME	Name of the file being opened.
	GS-CONTEXT-BUFFER	Blanks in the first four characters.
All other	FILE-NAME	Name of the file being accessed.
	FILE-NUMBER	The identifying number returned by the General Server following a successful open operation. File numbers are 0 through 31.
	RECORD-KEY (non-sequenced files only)	Primary key value.

Table E-3. Values Expected by the General Server for SEND Processing (Continued)

Operation	Field or Record	Value
	KEY-SPECIFIER (key-sequenced files only)	Two-byte specifier for the current key of reference (binary zero for the primary key), if access is by key.
	RECORD-LENGTH	The record length in bytes of the file being accessed or zero.
	RECORD-AREA-LENGTH	The General Server expects this value to be either the record length or zero. If the record length is an odd number, the General Server expects record length + 1.
Random READ	TRANS-CODE	01
	FUNCTION-CODE	EX READ EXACT.
		AP READ Approximate.
		GN READ GENERIC.
		AG READ Approximate Generic

Table E-3. Values Expected by the General Server for SEND Processing (Continued)

Operation	Field or Record	Value
	COMPARE-LENGTH	Byte length of the leading portion of the value to be used for positioning.
	RECORD-IMAGE	The key value in the appropriate position.
Sequential READ	TRANS-CODE	01
	FUNCTION-CODE	NX READ NEXT within current key set or within the file. NN READ NEXT establishing a new key set with approximate mode implied. FR READ FIRST record using record key specified.
	PRIOR-POSITION	The file position returned from the previous call to ENABLEGS (for NX and NN only).
	RECORD-IMAGE	The current record image (for NX and NN only).

Table E-3. Values Expected by the General Server for SEND Processing (Continued)

Operation	Field or Record	Value	
UPDATE	TRANSACTION-CODE	02	
	RECORD-IMAGE	The old record image.	
	RECORD-IMAGE-2	The new record image (must begin on a word boundary).	
DELETE	TRANSACTION-CODE	03	
	RECORD-IMAGE	The old record image.	
INSERT	TRANSACTION-CODE	04	
	FUNCTION-CODE	Blank	INSERT at first available space.
		EX	INSERT at indicated record key position. (relative files only).
		LA	INSERT at end of file (relative files only).
	RECORD-IMAGE	The new record image.	

GENERAL SERVER RESPONSE FORMAT

General Server responses consist of a fixed prefix common to all types of transactions and a variable portion dependent on the entry in REPLY-CODE. REPLY-CODE is equal to 0 or 1 as follows:

REPLY-CODE=0	Prefix only returned
REPLY-CODE=1	Prefix plus RECORD-IMAGE returned

The prefix consists of the 24-byte PATHWAY header, the 54-byte General Server header, and the 96-byte context buffer. The values returned in the prefix are the same as those passed in the request except as noted in the following paragraphs.

The following header fields are set by the General Server for the indicated REPLY-CODE and TRANS-CODE values of the request:

REPLY-CODE	All codes	
ERROR-STATUS	All codes	
FILE-NUMBER	TRANS-CODE=0	
RECORD-LENGTH	REPLY-CODE=0	All codes if zero passed in the request.
RECORD-AREA-LENGTH	REPLY-CODE=0	All codes if zero passed in the request.
FILE-POSITION	TRANS-CODE=1	

Normally the General Server returns:

- The RECORD-IMAGE in the variable portion of the General Server response for a successful READ operation. A successful READ operation is where the request has TRANS-CODE=1, and the response has ERROR-CODE=0 and REPLY-CODE=1.

- The prefix only when an operation is unsuccessful. An unsuccessful operation occurs when the request has any TRANS-CODE, and the response has a nonzero ERROR-CODE.

The only exception to this procedure concerns the handling of update conflicts, which occur when the present program attempts to modify or delete a record that was altered by another process since the present program read that record. If an UPDATE operation (TRANS-CODE=2) or DELETE operation (TRANS-CODE=3) fails because of update conflict, the new record image is returned in the response; REPLY-CODE is set to 1 and ERROR-CODE is set to a nonzero value.

GENERAL SERVER ERROR CODES

The General Server can return error codes during initialization and at run time.

During initialization, the General Server returns error codes as replies to system messages. If a fatal initialization condition occurs, the General Server calls ABEND.

At run time, the General Server returns errors in ERROR-CODE of the General Server header. Table E-4 lists the codes and their interpretations.

Table E-4. General Server Run-Time Errors

ERROR-CODE Value	Interpretation
1	The TRANS-CODE value was out of bounds.
2	The FUNCTION-CODE was unknown.
3	The data file identified in the request was not known or was not accessible.
4	The record could not be found. A random READ with an invalid key was attempted, a READ NEXT was beyond the current key set or file, or the record was locked.
5	An update conflict occurred on a DELETE or UPDATE operation. A program attempted to DELETE or UPDATE a record, but another process changed the record after the first program read it. The new record image is returned to the caller.
6	An invalid record length was entered. The length was not within the limits imposed by the file structure.
7	An invalid key ID was entered. The key was not one of the known keys.
8	The INSERT failed. See the FILE-ERROR-CODE.
9	The UPDATE failed. See the FILE-ERROR-CODE.
10	The DELETE failed. See the FILE-ERROR-CODE.

When the General Server receives an error issued by the file system, it returns the file-system error code in FILE-ERROR-CODE of the General Server header. The General Server reserves the ERROR-SUB-CODE field for codes to identify further the location in ENABLEGS at which the error condition was detected. The only meaningful ERROR-SUB-CODE is 96, which corresponds to the INSERT OK, but DELETE on old image failed error message.

When ERROR-CODE=3 is returned from an open transaction, additional information is returned in FILE-ERROR-CODE. Table E-5 shows the codes and their interpretations.

Table E-5. General Server File-Open Transaction Errors

FILE-ERROR-CODE Value	Interpretation
300	The number of keys specified by the open transaction does not match the number actually found in the file.
301	The record-length parameter was missing from the ASSIGN for the file.
302	TMF mode mismatch. The file is audited and the General Server is not running in TMF mode, or the file is not audited and the General Server is running in TMF mode.
303	The requester program version number is not compatible with the version number of the General Server. This happens if a requester is run against an old server.
304	The program no longer agrees with the data file. Different versions of DDL were used to generate the program and the data file.
305	NONSTOP and TMF parameters were both specified when opening the server.
306	Not enough memory remained in the General Server to support an OPEN by an additional requester.

APPENDIX F

ENABLE VERSIONS COMPARISON

This appendix describes the difference between ENABLE B00 (the version described in this manual) and ENABLE A00 (the previous version of ENABLE).

Table F-1 lists the differences between the terminal screens displayed by applications generated with ENABLE A00 and the terminal screens displayed by applications generated with ENABLE B00.

Table F-1. Application Display Screen Differences
(Continued next page)

ENABLE A00	ENABLE B00
The application displays a zero (0) in the Record Number field for the first record in a non-key-sequenced file.	The application displays a blank in the Record Number field for the first record of a non-key-sequenced file; the application equates the blank to a zero.
The application displays a deleted record after completing the delete operation.	The application does not display a deleted record after completing the delete operation.
The heading Page m/n, which indicates the current screen page, begins in the second column of the second screen row.	The heading Page m/n, which indicates the current screen page, begins in the first column of the second screen row.
The field for a Record Number begins one screen column after the label.	The field for a Record Number appears in the same screen column as the other fields from the record when SCREENFORMAT is UNCOMPRESSED.
The application sets error messages to blinking.	The application highlights error messages.
Highlight is reverse video.	Highlight is dim reverse video.
Long data fields wrap around screen lines. A wrapped line begins in column 1.	Long data fields wrap around screen lines and begin in column 1 for a box at the first level of the tree structure. The wrapped field begins in column 2 of a box that resides at a subsequent level of the tree.

Table F-1. Application Display Screen Differences (Continued)

ENABLE A00	ENABLE B00
The application checks a variable that shadows the Record Number field for relative, entry-sequenced, and unstructured files to see if this field has been altered by the user. The application does this for relative files to determine whether to insert the record at the specified location or in the first free location.	The application does not use another variable to shadow the Record Number field. Instead, the application compares the displayed and entered values of this field to determine whether the record number has been altered by the user.

Table F-2 compares the way that certain operations are performed by applications generated with both versions of ENABLE.

Table F-2. Application Operation Differences

ENABLE A00	ENABLE B00
The application does not support the NEXT PAGE and PREV PAGE keys (found only on T16-652x, and T16-653x Terminals).	The application supports the NEXT PAGE and PREV PAGE keys. They perform the same function as F2 and F1, respectively.
The application does not allow READ GENERIC operations on the Record Number field of relative, entry-sequenced, or unstructured files.	The application allows READ GENERIC operations on the Record Number field of relative, entry-sequenced, or unstructured files; however, the application translates the READ GENERIC operation into a READ EXACT.
The application allows you to insert a null (blank) record.	The application does not allow you to enter a record consisting of default values into a file with either an insert operation or an update operation. Default values consist of spaces or zeros if VALUES is OFF or initial values from the record description if VALUES is ON.
On IBM-327x terminals the UNDO and RECOVER SCREEN function keys are PF10 and PF12, respectively.	On the IBM-327x terminal the UNDO and RECOVER SCREEN function keys are shifted PF7 and PF9, respectively. These function keys have been changed so that the BOX operations (BOX INSERT, BOX DELETE, and BOX UPDATE) can be the shifted version of the corresponding operation.
On IBM-327x terminals, the CLEAR/DISPLAY function key is PA1.	On IBM-327x terminals, the CLEAR/DISPLAY function key is PF9.

Table F-3 lists generation discrepancies for the two versions of ENABLE.

Table F-3. Differences During Application Generation
(Continued next page)

ENABLE A00	ENABLE B00
The starting value of the VALUES attribute is ON.	The starting value for the VALUES attribute is OFF.
Valid values for the HEADINGS attribute are ON and OFF.	Valid values for the HEADINGS attribute are: DDLHEADINGS, DDLFIELDNAMES, and NULL. The values ON and OFF are still supported (DDLHEADINGS = ON, DDLFIELDNAMES = OFF).
The value of the DDLFIELDNAMES attribute determines whether ENABLE uses field names from the record description for data item names in the working storage area of the SCREEN COBOL program.	The DDLFIELDNAMES attribute is not supported. Working storage names always come from the field names in the record description. To maintain compatibility with ENABLE A00, you can supply a value for DDLFIELDNAMES without causing an error; however, no action will result.
ENABLE does not automatically set DELETE to off for unstructured or entry sequenced files; instead, ENABLE generates the application so that it displays the message "Operation not supported" when a delete operation is attempted.	ENABLE automatically sets DELETE to OFF for unstructured and entry-sequenced files.

Table F-3. Differences During Application Generation (Continued)

ENABLE A00	ENABLE B00
If SCOBOLOBJECT is null (indicating that compilation is not to take place), ENABLE issues the warning: "Generated 'SET TCLPROG' has no program file name."	IF SCOBOLOBJECT is null, ENABLE issues the message: "Generated Pathcomfile must be edited." ENABLE writes question marks to the PATHCOM command file to indicate where information is missing.
When ENABLE writes the PATHCOM command file, it uses minimum values for parameters. These values are sufficient to bring up a single application on a single terminal.	When ENABLE writes the PATHCOM command file, it uses values that provide more flexibility for the PATHWAY configuration.
In the PATHCOM command file, ENABLE uses the record description name as the logical file name for the SET SERVER ASSIGN command.	In the PATHCOM command file, ENABLE uses the box name as the logical file name for the SET SERVER ASSIGN command. For this reason, you should not try to use a PATHCOM command file generated by ENABLE A00 with an application generated by ENABLE B00.
The maximum number of nested OCCURS clause items that ENABLE can handle is 3.	The maximum number of nested OCCURS clause items that ENABLE can handle is 4 with one exception: if SIZE is greater than 1, ENABLE counts this as an OCCURS item.
After you generate SCREEN COBOL source code, you can freely change the location of the screen items.	After you generate SCREEN COBOL source code, you must be careful when moving screen items. You must keep each record contiguous and keep the first and last fields of each record in their respective positions.

Table F-3. Differences During Application Generation (Continued)

ENABLE A00	ENABLE B00
The VOLUME command alters both the volume and subvolume settings for CMDVOL and OBEYVOL when only a volume name is	When using the VOLUME, CMDVOL, and OBEYVOL commands, specifying a volume affects only the default volume setting; specifying a subvolume affects only the subvolume setting. To affect both settings, a volume and a subvolume must both be specified.
The CMDVOL command, with no arguments, results in an error.	The CMDVOL command, with no arguments, restores the CMDVOL file expansion default to its original value.

APPENDIX G

GLOSSARY

Access. The right of an application to open, read, or update information in a data base file.

Access path. An established order in which an application reads records.

Alphanumeric data. Data that consists of uppercase and lowercase letters of the alphabet, digits, blanks, and special characters.

Alternate key. A key field that identifies a record in a key-sequenced, entry-sequenced, or relative file; alternate keys need not have unique values.

APPL. A keyword that identifies the type of object that represents an application and its attributes.

Appl attribute. A characteristic of an application.

Application. A complete sequence of machine instructions and routines necessary to solve a problem.

Approximate mode. A positioning mode that provides record access by a key value equal to or greater than a supplied key value.

Assignment. A convention in which an ASSIGN command is issued to make logical file assignments for programs. A logical file assignment equates a Tandem file name with a logical file of a program and optionally attributes characteristics to that file.

Attribute. A characteristic of an object.

Attribute table. An internal table that ENABLE uses to store attribute values.

Audited file. A data file that is flagged for auditing by TMF; auditing is the monitoring of transactions in preparation for recovery efforts.

BOX. A keyword that identifies the type of object that represents a data base file and its attributes within an application.

Box. An element displayed by an application on the terminal screen. A box contains a record (or records) from a specific data base file.

Box attribute. A characteristic of a box.

Command interpreter. An interactive program used to run programs, check system status, create and delete disc files, and alter hardware states.

Command subvolume. The subvolume in effect when you enter the ENV operating command.

Command volume. The volume in effect when you enter the ENV operating command.

Composite key. A primary or alternate key field that consists of two or more contiguous fields.

Containing box. A box on the terminal screen within which another box is nested.

Courtesy key. The record number of an entry-sequenced, relative, or unstructured file.

Current attribute value. An attribute value supplied by a SET command.

Current record. The most recently retrieved record.

Cursor. A highlighted screen element that marks character position during terminal input.

Data administrator. An individual who is responsible for defining the format and organization of a data base.

Data Definition Language (DDL). A trademark that signifies the Tandem proprietary language used to describe the records and files composing a data base.

Data base. A collection of data that is described and controlled within a computer system.

Data dictionary. A set of files that provide information about each file in a data base.

Data type. A category that identifies the kind of data that a field represents. Four broad categories of data types exist: alphabetic, alphanumeric, numeric integer, and numeric noninteger.

Data values. The actual values stored in a data base file.

Default attribute value. An attribute value supplied by ENABLE when both the starting and current attribute values are null.

Default object type. A object type that ENABLE uses when you omit the keyword BOX or APPL from an ENABLE command. The ASSUME command affects the default object type.

Default value. A value that is used by the system when a value has not been supplied by the user.

Edit-type file. A source text file that can be augmented and modified by the user through a text editor.

ENABLE. A trademark that signifies the Tandem proprietary application-generation subsystem.

ENABLE commands. Commands that are associated with use of the ENABLE subsystem.

ENSCRIBE. A trademark that signifies the Tandem proprietary data base record manager.

Entry-sequenced file. A file in which records are stored in the order in which they are written into the file. Records can be identified by a record number that indicates the position of the record within the file.

Exact mode. A positioning mode that provides record access by a key value exactly matching a supplied key value.

Field. An element that represents the storage area for one specific group of letters, numbers, or letters and numbers.

File. A collection of records.

File Utility Program (FUP). A trademark that signifies the Tandem proprietary utility program that is used for performing certain disc file related operations.

General Server. The Tandem proprietary process, supplied by the ENABLE subsystem, that provides access and updates data base files.

Generic mode. A positioning mode that provides record access by a key value matching a supplied partial key value.

Group. A field in a record description made up of two or more contiguous elementary fields.

GUARDIAN. The Tandem operating system.

Heading. A name established in the DDL dictionary that can replace the field name on reports or on the screen.

Interactive mode. An operating mode in which commands are entered from a terminal keyboard.

Join field. A field from one box that matches a field from another box. Fields match if they have compatible data types and represent common data values.

Key field. A field, the value of which is used to identify a specific record within a file.

Key of reference. Either the primary key or alternate key currently being used to access a record.

Key-sequenced file. A file in which records are stored in ascending sequence according to the value of the primary key field.

Link. A logical connection between the boxes used by an application.

Linked field. A join field from a child box.

Linking field. A join field from a parent box.

Multifile application. An application generated by ENABLE that can access two or more data base files or a single data file opened as two or more data files.

Nested box. A box on the terminal screen that is contained by another box.

Noninteractive mode. An operating mode in which commands are entered through a command file.

NonStop. A trademark signifying the failure-tolerant features of the proprietary Tandem architecture and operating system.

NONSTOP. An ENABLE attribute used to specify whether the General Server is to operate as a NonStop process pair.

Numeric data. Data that consists of digits (0-9); leading and trailing blanks; and possibly a decimal point and a minus sign.

Obey file. A file that serves as an alternate source for command input.

Object. An application or a box. An object table entry that describes an application or a box.

Object table. An internal table that ENABLE uses to store information from which applications are generated.

Object type. An entity that can be the subject of a SET command. ENABLE currently supports two object types: APPL and BOX.

Operating commands. Commands that are associated with control of the ENABLE program.

Operation. An act performed by an application upon a data base file.

Outermost box. The highest level box in a multifile application. The terminal screen itself forms the box that displays the screen label and field pairs for the outermost box.

Override attribute value. A temporary attribute value supplied by an ADD command; the value only applies to the object being added.

PATHCOM command file. A file of commands that define PATHWAY objects required to execute an application.

PATHCTL. A disc file in which PATHMON maintains status information and the application configuration.

PATHMON. The central controlling process in a PATHWAY system.

PATHWAY. A trademark signifying the Tandem proprietary transaction processing system that supplies the programs, procedures, and structures necessary to execute user-written applications.

Positioning Mode. One of three modes that establish a subset of records in a designated access path: approximate, exact, and generic.

Primary key. The key field that uniquely identifies a record in a file; a primary key cannot be duplicated.

Program generator. The component of ENABLE that generates SCREEN COBOL source code.

Record. Depending on the context in which it is used, a record is either related data stored in a data base file or a record description.

Record description. A entity stored in a data dictionary that describes the organization and structure of a data base file.

Record number. An ordinal value that uniquely identifies a record in an entry-sequenced, relative, or unstructured file.

Relative file. A file in which records are stored in a position relative to the beginning of the file. Records within the file can be identified by a record number.

Requester process. A process that interprets application-program object code and sends replies to a server; synonymous with requester.

SCREEN COBOL. A trademark that signifies the Tandem proprietary procedural language for terminal display control under PATHWAY.

Server. A process that handles file I/O processing under PATHWAY.

Skeleton file. A file of SCREEN COBOL source text or PATHCOM commands, plus special commands that drive ENABLE processing; the file can be used in its present state or changed by the application programmer.

Single-file application. An application generated by ENABLE that can access a single data base file.

Spooler. A process that serves as a buffer between a print device and an application writing to the device.

Starting value. An attribute value that exists when you start ENABLE.

Subset. A related set of records in an access path.

Subsystem. A program that is supplied as part of the operating software.

Sync ID. A value used by the operating system to provide automatic path error recovery for disc files.

TCP. A program supplied by Tandem that interprets SCREEN COBOL object code and sends messages to server processes; synonymous with requester process.

Terminal. A device capable of sending and receiving information over communication lines.

Transaction Monitoring Facility (TMF). A trademark that signifies the Tandem proprietary data management product that monitors a data base for consistency and provides the tools for data base recovery.

Tree structure. A logical structure that ENABLE uses to identify the boxes that are associated with an application. For a multifile application, a tree structure also identifies the links that exist between the boxes and the order in which the boxes are linked.

Unstructured file. A file in which data is physically located in 512-byte sectors and is referred to by a relative byte address.

INDEX