Data Management Library

ENABLE™ User's Guide

Abstract This guide describes how to use ENABLE to generate a variety of data base management applications.

Part Number 82571 A00

Tandem Computers Incorporated

Document History

Edition	Part Number	Product Version	Operating System Version	Date
First Edition	82571 A00	ENABLE B00	GUARDIAN B00	March 1985

New editions incorporate any updates issued since the previous edition.

CONTENTS

FIGURES

PREFACE

This guide describes how to use ENABLE, a product that is part of the ENCOMPASS distributed data base management system provided by TANDEM, to build simple transaction processing applications.

To use this guide, you must understand your information needs, have a working knowledge of the Tandem text editor, and be familiar with some data management application programs. As an ENABLE user, you need not know the SCREEN COBOL or COBOL programming languages.

The guide is organized in a sequence that parallels the development of an ENABLE application:

- Section 1 introduces the capabilities and uses of ENABLE and identifies its components.

- Section 2 provides an overview of the steps required to produce an application. This section also introduces some concepts referred to throughout the remainder of the guide.

- Section 3 provides guidelines for developing, describing, and creating a data base.

- Section 4 describes the tasks you must perform to develop an application that can access a single data base file.

- Section 5 describes the tasks you must perform to develop an application that can access more than one data base file.

- Section 6 describes methods you can use to tailor an ENABLE application.

- Section 7 provides guidelines for modifying the SCREEN COBOL source code generated by ENABLE

- Section 8 provides guidelines for using an ENABLE application.

- Section 9 provides guidelines for resolving problems encountered during application generation or execution.
- Section 10 provides guidelines for maintaining applications generated by ENABLE.
- Section 11 describes the tasks you must perform to integrate several ENABLE applications into a single PATHWAY system.
- Section 12 provides an example of an integrated system with several ENABLE applications.

First time users of ENABLE should read sections 1, 2, 3, 4, 5, and 6 in that order. If a data dictionary and data base files already exist, you can omit Section 3.

Individuals who have used the previous version of ENABLE should read sections 1, 2, 5, and 6. If you must create a data dictionary or data base files, read section 3.

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

- <u>Data Definition Language (DDL) Reference Manual</u>
- <u>ENABLE Reference Manual</u>
- <u>GUARDIAN Operating System User's Guide</u>
- <u>PATHWAY System Management Reference Manual</u>
- <u>PATHWAY SCREEN COBOL Reference Manual</u>
- <u>PATHAID Reference Manual</u>

When you use an application developed by ENABLE, you should have the appropriate ENABLE template. These templates and the corresponding terminal type are as follows:

• Part Number 82174 B00	For T16-651x and T16-652x terminals
• Part Number 45637	For T16-653x terminals

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.

SECTION 1

INTRODUCTION TO ENABLE

Many companies have a backlog of programming projects requested by their departments. While these projects are important to the department involved, most companies use their data processing resources to develop projects that are important to the organization as a whole; projects for a single department must often wait.

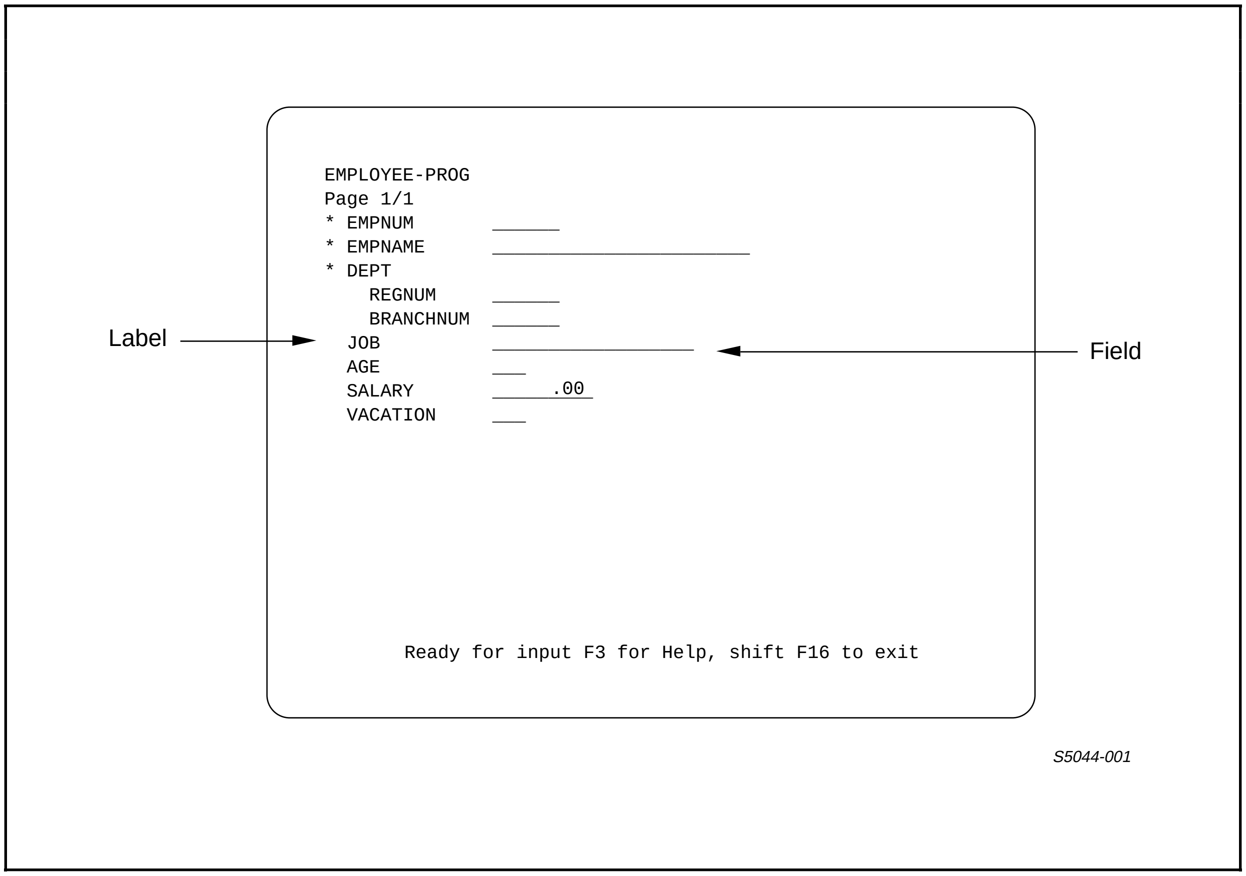
If your department needs a data-management application, ENABLE may offer a solution. You can use ENABLE to develop an application that can record, maintain, or retrieve information stored within a data base. ENABLE is a powerful tool that allows you to develop simple data-management applications, without using a conventional programming language.

Although it might not solve your problem with as much sophistication as a custom-designed application program, you can quickly generate an ENABLE application to meet your immediate needs.

WHAT CAN AN ENABLE APPLICATION DO?

An ENABLE application can enter, retrieve, and display information from one or more related data base files. The application performs these operations on a record-by-record basis and on one file at a time.

ENABLE generates online applications that you use interactively through a terminal. To simplify the process of entering and changing information, they display a form on the screen similar to the one shown in Figure 1-1. The application that displays this form can retrieve and enter information for one data base file.

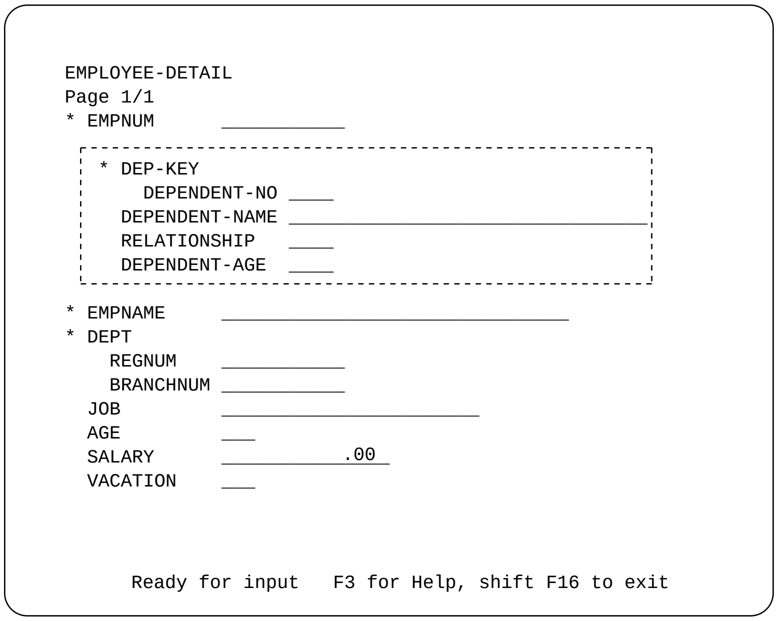


```
EMPLOYEE-PROG
Page 1/1
* EMPNUM       ______
* EMPNAME      _______________________
* DEPT
    REGNUM     ______
    BRANCHNUM  ______
  JOB          __________________
  AGE          ___
  SALARY       _____.00
  VACATION     ___

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

Figure 1-1. Sample Screen Displayed by a Single-File Application

Notice that the application provides labels for each piece of information that appears on the screen. The application also provides fields that display information and that you can use to enter information. Because applications similiar to this application access information from only one data base file, these applications are sometimes called single-file applications.

Figure 1-2 shows a form displayed by another ENABLE application. This application can process simple transactions for two data base files.



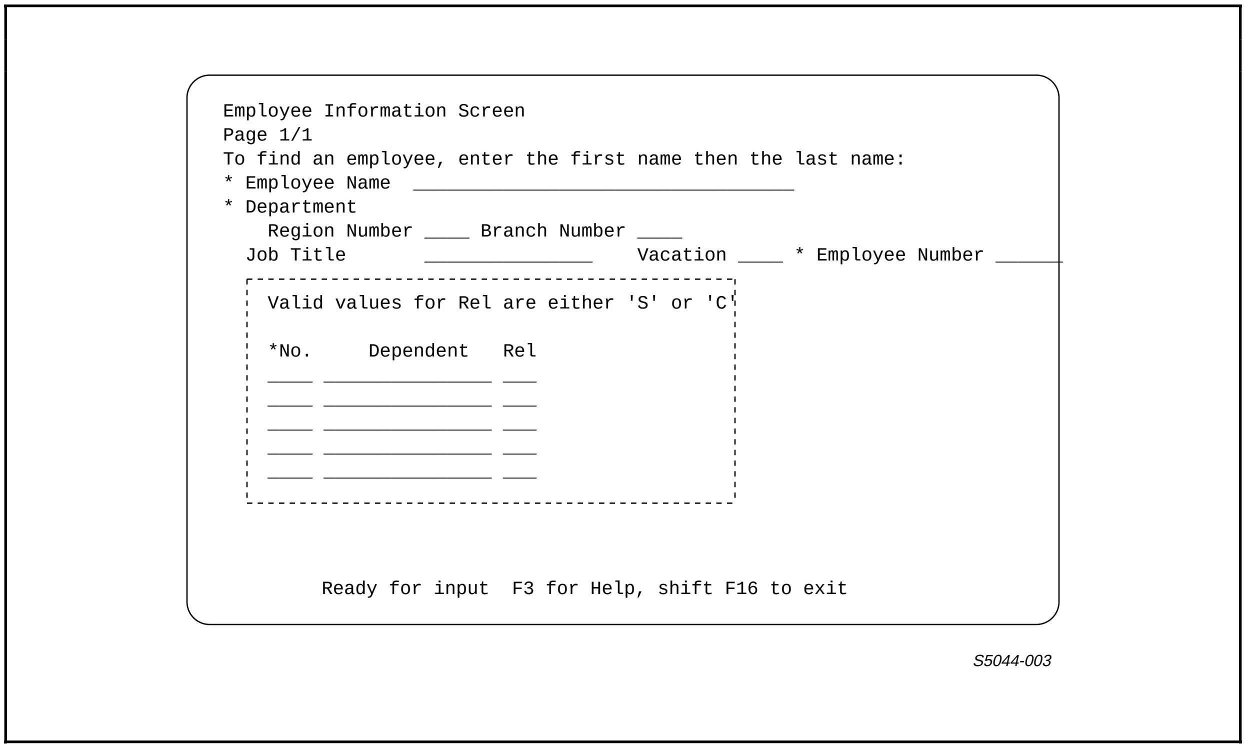
```
EMPLOYEE-DETAIL
Page 1/1
* EMPNUM      ___________

 * DEP-KEY
     DEPENDENT-NO ____
   DEPENDENT-NAME ________________________________
   RELATIONSHIP   ____
   DEPENDENT-AGE  ____

* EMPNAME     ________________________________
* DEPT
    REGNUM    ___________
    BRANCHNUM ___________
  JOB         ________________________
  AGE         ___
  SALARY      ___________.00_
  VACATION    ___


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

S5044-002

Figure 1-2. Sample Screen Displayed by a Multifile Application

Like the application whose screen appears in Figure 1-1, this application provides labels and fields. To distinguish between a record from one data base file and a record from the other file, the application encloses some of the labels and fields within a box. Because applications like this one can access information from more than one data base file, these applications are sometimes called multifile applications.

When you develop an application, ENABLE provides you with the ability to tailor specific areas of the screen. Figure 1-3 shows a screen that has been tailored so that records appear in a tabular format.



```
Employee Information Screen
Page 1/1
To find an employee, enter the first name then the last name:
* Employee Name  __________________________________
* Department
    Region Number ____ Branch Number ____
  Job Title       _______________    Vacation ____ * Employee Number ______

    Valid values for Rel are either 'S' or 'C'

    *No.     Dependent   Rel
    ____ _______________ ___
    ____ _______________ ___
    ____ _______________ ___
    ____ _______________ ___
    ____ _______________ ___


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

Figure 1-3. Sample Screen Displaying Several Records in a Tabular Format

The application that produces this screen can display more than one record within the box.

TYPICAL TASKS PERFORMED BY AN ENABLE APPLICATION

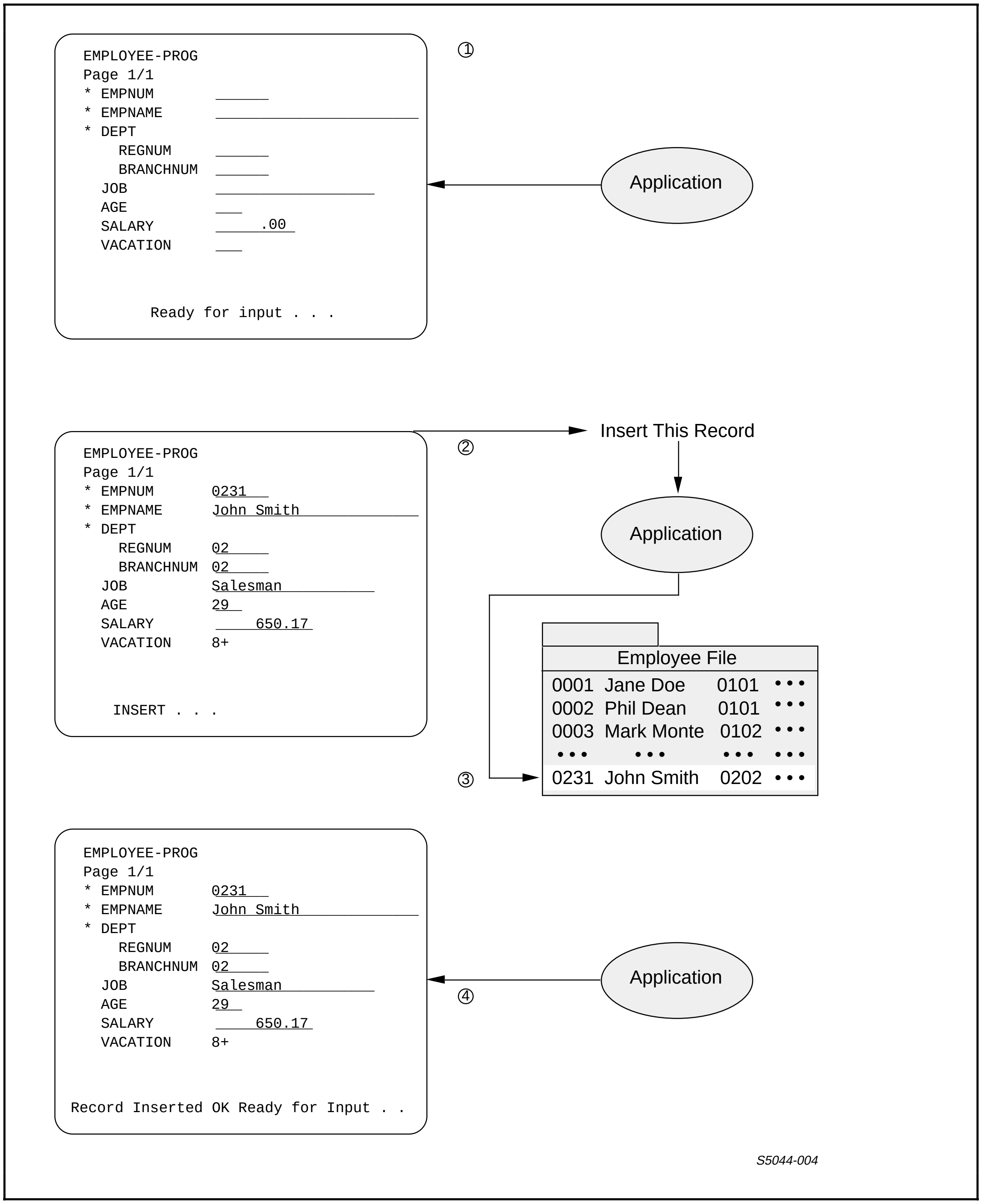
Figure 1-4 illustrates some typical tasks that can be performed by an ENABLE application. The application shown in this figure maintains information in an employee file. The application repeats the following tasks until the user exits:

1. Displays the screen; the user fills in the necessary information

2. Accepts a request from the user to insert the entered information

3. Accesses the data base to insert the information

4. Tells the user that the request has been successfully processed and that it is ready to accept another request



```
EMPLOYEE-PROG
Page 1/1
* EMPNUM        ______
* EMPNAME       _______________________
* DEPT
    REGNUM      ______
    BRANCHNUM   ______
  JOB           __________________
  AGE           ___
  SALARY        _____.00_
  VACATION      ___

        Ready for input . . .
```

```
EMPLOYEE-PROG
Page 1/1
* EMPNUM        0231__
* EMPNAME       John Smith_____________
* DEPT
    REGNUM      02____
    BRANCHNUM   02____
  JOB           Salesman__________
  AGE           29_
  SALARY        ____650.17_
  VACATION      8+

    INSERT . . .
```

Employee File			
0001	Jane Doe	0101	• • •
0002	Phil Dean	0101	• • •
0003	Mark Monte	0102	• • •
• • •	• • •	• • •	• • •
0231	John Smith	0202	• • •

```
EMPLOYEE-PROG
Page 1/1
* EMPNUM        0231__
* EMPNAME       John Smith_____________
* DEPT
    REGNUM      02____
    BRANCHNUM   02____
  JOB           Salesman__________
  AGE           29_
  SALARY        ____650.17_
  VACATION      8+

Record Inserted OK Ready for Input . .
```

Figure 1-4. Typical Tasks Performed by an ENABLE Application

The preceding tasks are especially typical of transaction-processing applications that keep information current and correct by making immediate (instead of deferred) changes to the data base.

TANDEM PRODUCTS AND ENABLE APPLICATIONS

As in any PATHWAY application, the tasks performed by an ENABLE application are divided logically among the following components:

1. A request-oriented program (requester) that displays the data entry screen, accepts the data entered from the terminal, and passes the data to programs that update the data base

2. A data base service program (server) that actually records, modifies, and retrieves information from the data base

ENABLE generates a SCREEN COBOL requester program that defines terminal display screens and accepts requests. ENABLE supplies a server program, called the General Server, that accesses the data base and performs the requested operation. Optionally, ENABLE also produces a third component, a command file that you can use to execute the application under a PATHWAY system.

PATHWAY, a Tandem product, controls online transaction processing applications. PATHWAY supplies a monitor process (PATHMON) that controls the PATHWAY system and a command interface (PATHCOM) that uses the command file, called a PATHCOM command file, produced by ENABLE. By entering a few commands, you can establish a PATHWAY system and execute an ENABLE application. Sections 2, 4, and 5 provide more information about PATHWAY.

After you use an ENABLE application to enter or modify information in your data base, you can use another Tandem product, ENFORM, to produce reports based on the data in your data base.

WHY USE ENABLE?

ENABLE reduces the amount of time needed to develop a simple application, thereby decreasing application development costs. To write a simple application in a conventional programming language could take hours, days, or even weeks of work. If you use ENABLE to generate the same application, you can be finished in a matter of minutes. ENABLE allows you to:

- Control the format of the screen displayed by the application

- Limit the types of operations (delete, insert, read, or update) that the application can perform on a data base file

- Define a method that the application uses to ensure the integrity of a data base file

Although ENABLE provides you with enough flexibility to tailor a simple application, an ENABLE application is not suitable for every purpose. An ENABLE application, for example, can neither perform mathematical calculations nor ensure the consistency of your data base files. ENABLE does, however, provide several options that allow you to modify an application and supply your own calculations or integrity constraints.

A standard ENABLE application (one that has not been modified) can be used as:

- A prototype of a more complex application--An analyst, for example, might use an application generated by ENABLE as a basis for determining end-user needs.

- A data entry program--An application generated by ENABLE can be used to enter information into a data base that will ultimately be maintained by a more complex application.

- A tool to maintain a personal data base--Any individual can use an application generated by ENABLE to maintain their personal data base files.

If you modify an ENABLE application, it can be:

- Integrated into an existing PATHWAY system--Within the PATHWAY system, you could use the application to list information from one or more data base files or to retrieve information used for another component of the PATHWAY system.

- Used to maintain a small independent data base--Because a standard ENABLE application typically does not maintain all desired integrity constraints, you should modify the application before you use it for a data base that requires these constraints.

HOW DOES ENABLE WORK?

When you use ENABLE to generate an application, ENABLE performs the following tasks:

1. Accepts specifications that describe your application

2. Opens a data dictionary and obtains a description of the files to be accessed by your application

3. Stores information about the applications to be generated in an internal table called the object table

4. Uses the information in the object table to:

   --Transform a SCREEN COBOL skeleton file into SCREEN COBOL source code

   --Transform a PATHCOM skeleton file into a PATHCOM command file

5. Calls the SCREEN COBOL compiler (SCOBOLX) to compile the generated SCREEN COBOL source code

Figure 1-5 illustrates these tasks.

ENABLE supplies the SCREEN COBOL skeleton file and the PATHCOM skeleton file. ENABLE allows you to modify these files before you generate an application.

You must supply a data dictionary and specifications that describe your application. Refer to Section 2 for information about the tasks that you must perform to generate and execute an application.

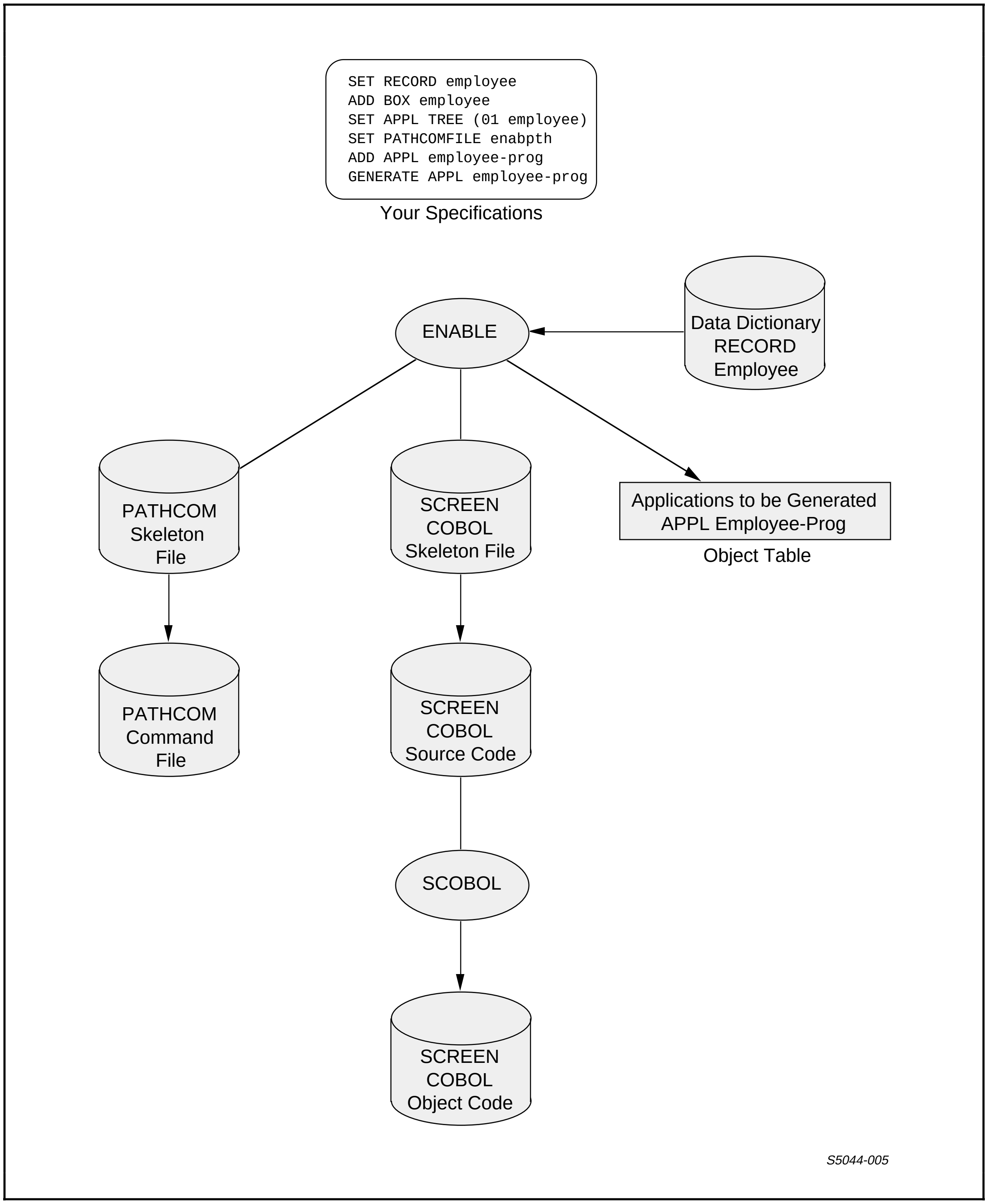


Figure 1-5. Tasks Performed by ENABLE

SECTION 2

GENERATING AND EXECUTING AN APPLICATION--OVERVIEW

This section provides an overview of the tasks that you perform to generate and execute an ENABLE application. These tasks are:

1. Providing a data dictionary that contains a description of each file to be accessed by the application

2. Supplying specifications, in the form of ENABLE commands, that describe the application

3. Creating the data base file (or files) to be accessed by the application if the files do not already exist

4. Establishing a PATHWAY system to execute the application

PROVIDING A DATA DICTIONARY

A data dictionary is a set of files that document the structure and organization of a data base. A data dictionary contains one or more record descriptions; each record description provides a format for the records stored in a data base file.

If a dictionary does not exist, you must create one before you try to generate an application. Section 3 provides information about creating a dictionary.

If a dictionary exists but does not contain an appropriate record description, you must add the record description to it before you try to generate an application. Section 3 also contains information about adding a record description to an existing dictionary.

SUPPLYING SPECIFICATIONS

When you use ENABLE, you must supply specifications that define and control program components called objects. There are two types of objects recognized by ENABLE: applications and boxes.

An application consists of a SCREEN COBOL program generated by ENABLE, the optional PATHCOM command file, and the provided General Server. Each application has attributes that describe it and for which you can supply values. For example, you can supply a value such as T16-6520 for the attribute (TERMINAL) that describes the type of terminal upon which an application runs. By supplying values for other application attributes, you can describe such global characteristics of an application as:

- The name of the file to which ENABLE writes the PATHCOM commands

- The title that the application displays on the terminal screen

A box represents a particular data base file and how it is to be handled by the application. The term "box" is derived from the box that a multifile application uses to enclose data from a file. Each box has attributes that describe it and for which you can supply values. You can, for example, indicate:

- How the application is to display information from the file on the teminal screen

- The types of operations (DELETE, INSERT, READ, or UPDATE) that the application can perform upon the file

You define and control these applications and boxes by using the ENABLE commands listed in Table 2-1. Refer to the ENABLE Reference Manual for detailed information about these commands. Each command has an optional keyword that indicates the type of object to which it applies. These keywords are:

- APPL--to indicate that the command applies to an application
- BOX--to indicate that the command applies to a box

Table 2-1. Summary of ENABLE Commands

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

When you want to describe an application, you can supply a value for one or more of the attributes listed in Table 2-2. Note that ENABLE supplies starting values (values supplied when you start ENABLE) or default values (values supplied when you add an application) for most application attributes. This means that you can provide a complete description of an application without supplying a value for every application attribute.

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

Attribute	Characteristic	Value Required?
PATHCOMFILE	Identifies the name of the PATHCOM command file. Starting value: null (no name)	No; however, ENABLE will not produce a PATHCOM command file if you do not supply a value.
PATHCOMSKELETON	Identifies the name of the PATHCOM skeleton file. Starting value: ENABPATS	No
SCOBOLCOMPILER	Identifies the SCREEN COBOL compiler that produces the SCREEN COBOL object code. Starting value: $system.system.scobolx	No
SCOBOLLIST	Identifies a file to which the SCREEN COBOL compilation listing is written. Starting value: null (no file)	No
SCOBOLOBJECT	Identifies the name of the object files for the compiled SCREEN COBOL program. Starting value: POBJ	No

Table 2-2. Summary of Application Attributes (Continued)

Attribute	Characteristic	Value Required?
SCOBOLSKELETON	Identifies the name of the SCREEN COBOL skeleton file. Starting value: ENABAPPS	No
SCOBOLSOURCE	Identifies the file to which the SCREEN COBOL source code is written. Starting value: null (no file)	No
TERMINAL	Identifies the type of terminal on which the application will run. Starting value: the type of terminal from which you enter ENABLE commands	No
TITLE	Identifies text that is displayed on the first screen line by the application. Starting value: null (blank); defaults to the application name	No
TREE	Identifies the box (or boxes) used by an application. Starting value: null (blank); defaults to: (01 <appl-name>-BOX)	Required for multifile applications; recommended for single-file applications.

When you want to describe a box, you can supply a value for one or more of the attributes listed in Table 2-3. Note that ENABLE supplies starting values (values that exist when you start ENABLE) or default values (values that exist when you add a box) for most box attributes. This means that you can provide a complete description of a box without supplying a value for every box attribute.

Table 2-3. Summary of Box Attributes (Continued next page)

Attribute	Characteristic	Value Required?
BOXTITLE 1 BOXTITLE 2 BOXTITLE 3	Identifies text that appears on the terminal screen with the records from a file. Starting value: null (blank)	No
CHECKDATA	Indicates whether the application contains special code that verifies the contents of numeric fields in a file. Starting value: ON (numeric fields are verified)	No
DATAFILE	Identifies the name of a data base file. Starting value: null; defaults to the file named in the DDL record description	No

Table 2-3. Summary of Box Attributes (Continued)

Attribute	Characteristic	Value Required?
DELETE	Indicates whether the application can delete records from a file. Starting value: ON (the application can delete records from the file)	No
DICTIONARY	Identifies the location of the data dictionary that contains the record description of a file. Starting value: the system, volume and subvolume from which you enter ENABLE commands	No
EXCLUDE	Identifies fields from the record description that are not to appear on the terminal screen. Starting value: Null (no fields are excluded)	No

Table 2-3. Summary of Box Attributes (Continued)

Attribute	Characteristic	Value Required?
FILL	Indicates whether the application performs an automatic read operation on a file under certain conditions. Starting value: OFF (an automatic read is not performed)	No
FLAG	Sets values for user flags defined in the SCREEN COBOL skeleton. Starting value: 0	No
HEADINGS	Identifies the set of labels to be used on the screen. Starting value: DDLFIELDNAMES (field names appear as labels)	No
INCLUDE	Identifies fields from the record description that are to appear on the screen; also indicates the order in which the fields appear. Starting value: null (all fields appear in DDL record-description order)	No

Table 2-3. Summary of Box Attributes (Continued)

Attribute	Characteristic	Value Required?
INSERT	Indicates whether the application can insert records in a file. Starting value: ON (records can be inserted)	No
NONSTOP	Indicates whether the General Server runs as a NonStop process pair. Starting value: OFF (the General Server does not run as a NonStop process pair)	No
READ	Indicates whether the application can, or cannot, read records from a file. Starting value: ON (records can be read)	No
RECORD	Identifies the name of the record description that describes a file. Starting value: null	Yes
SCREENFORMAT	Identifies the choice of screen layout for a file. Starting value: UNCOMPRESSED (one label and field appear on a screen line)	No

Table 2-3. Summary of Box Attributes (Continued)

Attribute	Characteristic	Value Required?
SERVERCLASS	Provides a name for the server class to which the General Server belongs. Starting value: ENABLE SERVER	No
SIZE	Identifies the number of records from a file that the application can display at one time. Starting value: 1	No
TMF	Indicates whether the box is to access a file audited by the Transaction Monitoring Facility (TMF). Starting value: OFF (the file is not audited by TMF)	No
UPDATE	Indicates whether the application can update records in a file. Starting value: ON (records can be updated)	No

Table 2-3. Summary of Box Attributes (Continued)

Attribute	Characteristic	Value Required?
VALUES	Indicates whether the application does, displays initial field values from the DDL record description. Starting value: OFF (initial values do not appear)	No

Supplying Values for Box and Application Attributes

When you supply a value for an application or box attribute, ENABLE stores the value in an internal table called the attribute table. To ensure that an object is adequately described, ENABLE stores the following in the attribute table:

- The starting value of each attribute--a value that an attribute has when you start ENABLE

- The current value (if any) of an attribute--a value that you supply with a SET command

  When you supply a current value for an attribute, the current value remains in the attribute table until you supply a new current value (with another SET command), remove the current value (with a RESET command), or exit from ENABLE.

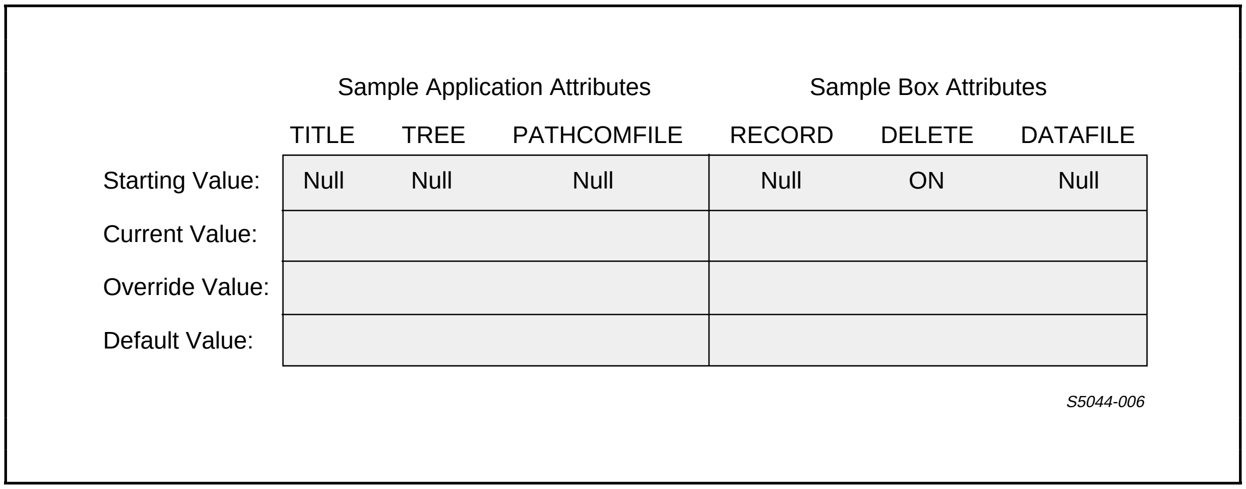
- The override value (if any) of an attribute--a value that you supply with an ADD command

  When you supply an override value that value only applies to the object being named and added with the ADD command.

- A default value for some attributes--a value that ENABLE supplies for some attributes with null starting values when you do not supply a current or override value.

  ENABLE supplies a default value when an object is added. A default value only applies to the added object.

Figure 2-1 illustrates the partial contents of the attribute table when you start ENABLE.



	Sample Application Attributes			Sample Box Attributes		
	TITLE	TREE	PATHCOMFILE	RECORD	DELETE	DATAFILE
Starting Value:	Null	Null	Null	Null	ON	Null
Current Value:						
Override Value:						
Default Value:						

S5044-006

Figure 2-1. Attribute Table When You Start ENABLE

When ENABLE adds an object to the object table, it uses the contents of the attribute table to determine the description of the object. ENABLE evaluates the contents of the attribute table as follows:

1. Attributes for which you have supplied override values--ENABLE uses these values to determine the object's description.

2. Attributes for which you have not supplied override values, but for which you have supplied current values--ENABLE uses the the current values to determine the object's description.

3. Attributes for which you have supplied neither current nor override values--ENABLE uses the starting values to determine the object's description.

4. Certain attributes whose starting values are null and for which you have supplied neither override nor current values--ENABLE supplies default values to determine the object's description.

Effect of the ENABLE Commands

Typically, you use the ENABLE commands to perform the following tasks:

1. Describe a box by supplying values for the box attributes.
2. Name the box and add it to the object table.
3. Repeat steps 1 and 2 until you have described and added every box to be used by the application.
4. Describe the application by supplying values for the application attributes.
5. Name the application and add it to the object table.
6. Generate the application.

The following paragraphs describe the action that ENABLE takes when you perform the preceding tasks to:

- Provide a complete description of a box named "employ-box"
- Add this box to the object table
- Describe an application named "employ-in"
- Add this application to the object table
- Generate "employ-in"

Describing and Adding a Box

The values of the box attributes describe a box. To supply a value for these attributes, you can use one of the following commands:

- The SET command--which supplies a current value for the attribute

- The ADD command--which supplies an override value for the attribute

To identify the record description for "employ-box," for example, you could use the following SET command to supply a value for the RECORD attribute:

```
   SET BOX RECORD employee
```

Note that you do not have to include the keyword BOX when you use the SET command to supply a value for a box attribute. To maintain compatibility with a previous version of ENABLE, ENABLE allows you use the SET command to supply a value for a box attribute without including this keyword; for example, the command:

```
   SET RECORD employee
```

could also be used.

Figure 2-2 illustrates the action ENABLE takes if you enter the preceding SET command.

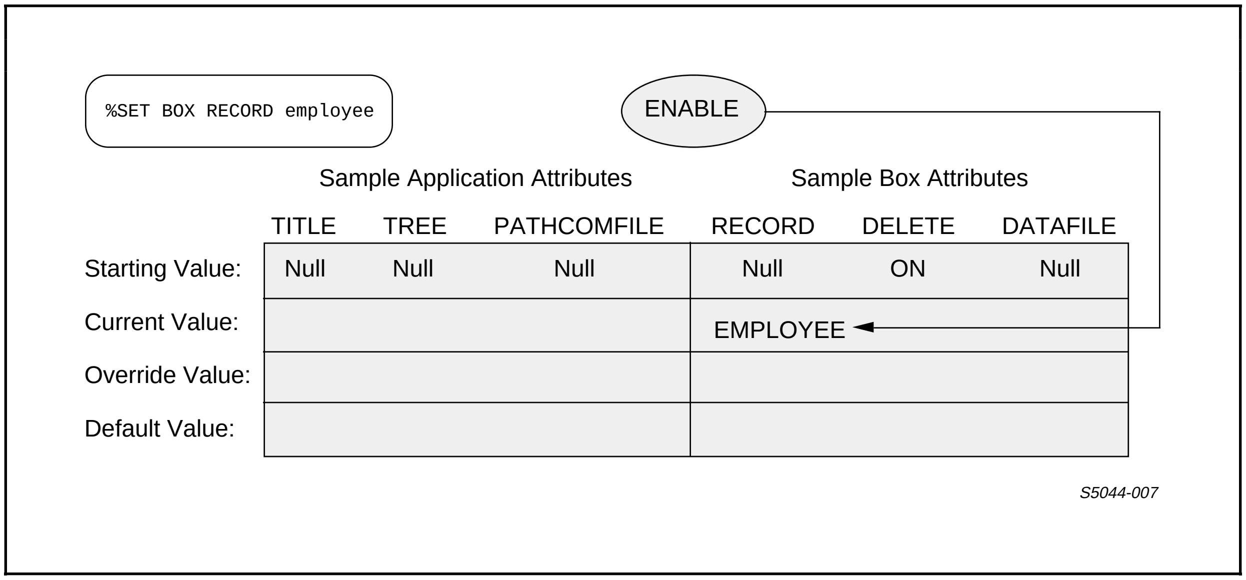


Figure 2-2. Attribute Table With a Current Box-Attribute Value

You could supply OFF as an override value for the DELETE attribute of "employ-box" by entering the following command:

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

Figure 2-3 illustrates the effect of the preceding ADD command.

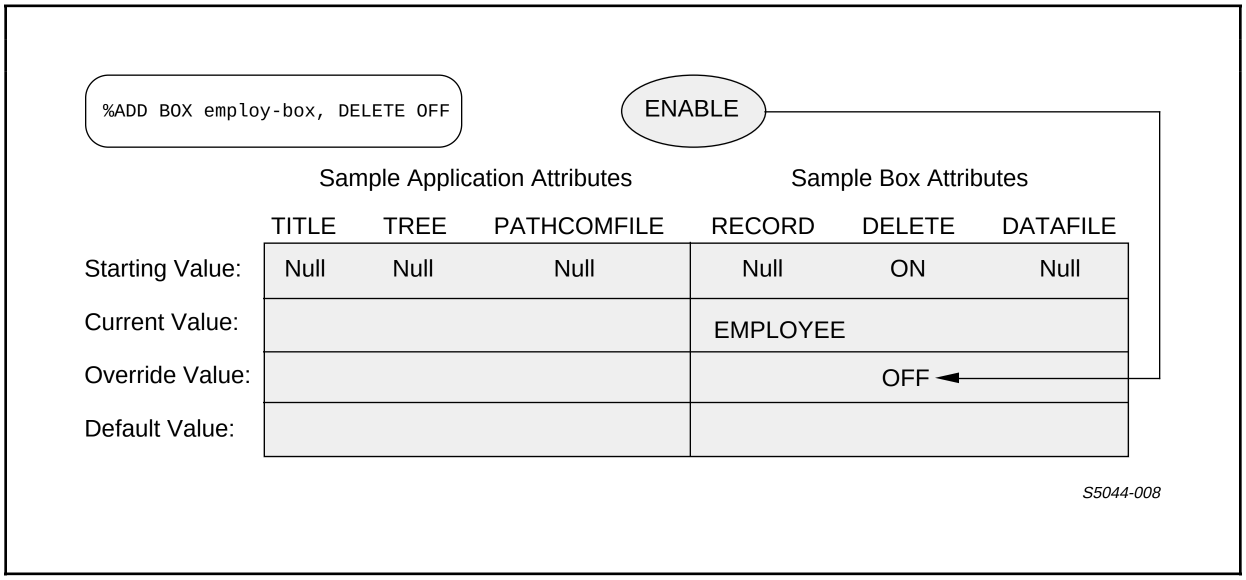
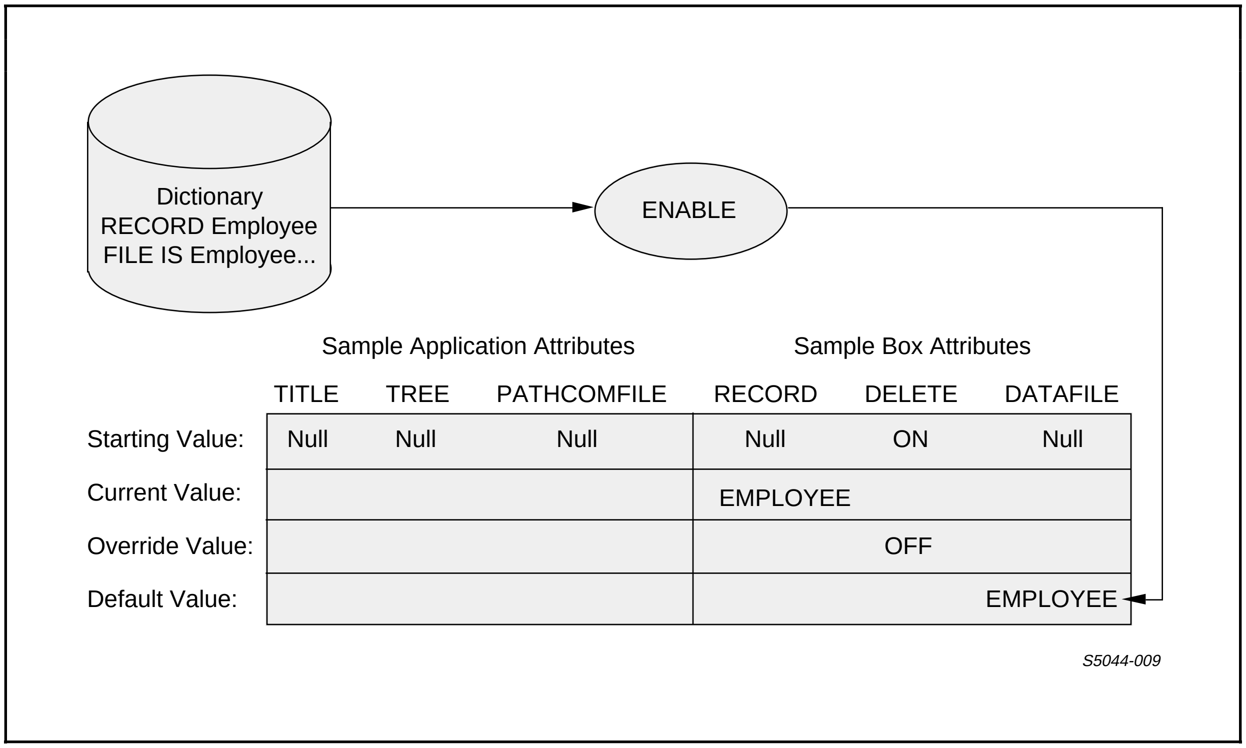


Figure 2-3. Attribute Table With an Override Box-Attribute Value

After you enter the ADD BOX command, ENABLE examines the contents of the attribute table to obtain the description of the box. If certain attributes retain a null starting value (you have not supplied a current or override value), ENABLE supplies a default value. For example, the starting value of the DATAFILE attribute in Figure 2-3 is null and the attribute table contains neither an override nor current value for this attribute. Since this attribute has a default value, ENABLE will supply "employee" as the default value of the DATAFILE attribute for "employ-box."

Figure 2-4 illustrates the partial contents of the attribute table when ENABLE adds the default value of the DATAFILE attribute. ENABLE obtains this default value from the record description in the dictionary.



	Sample Application Attributes			Sample Box Attributes		
	TITLE	TREE	PATHCOMFILE	RECORD	DELETE	DATAFILE
Starting Value:	Null	Null	Null	Null	ON	Null
Current Value:				EMPLOYEE		
Override Value:					OFF	
Default Value:						EMPLOYEE

Figure 2-4. Attribute Table With a Default Box-Attribute Value

After entering any default values in the attribute table, ENABLE adds "employ-box" and its description to the object table, as Figure 2-5 illustrates.

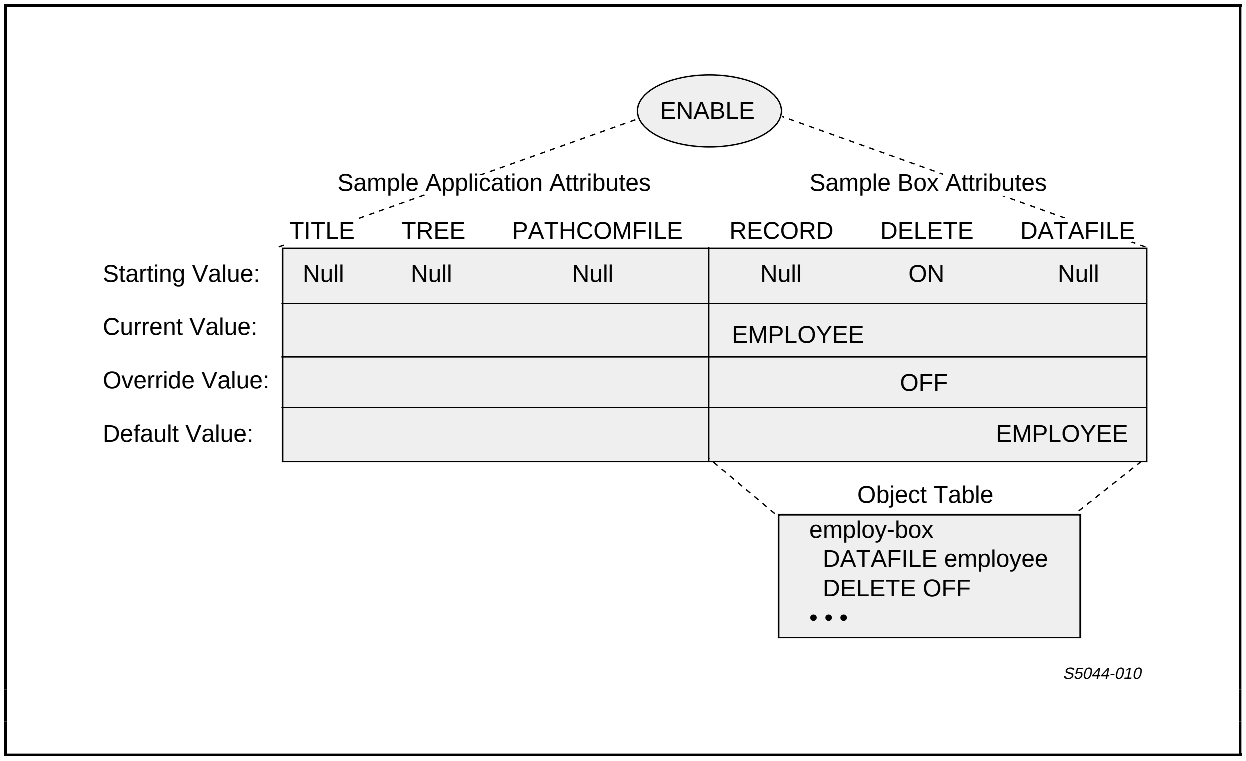
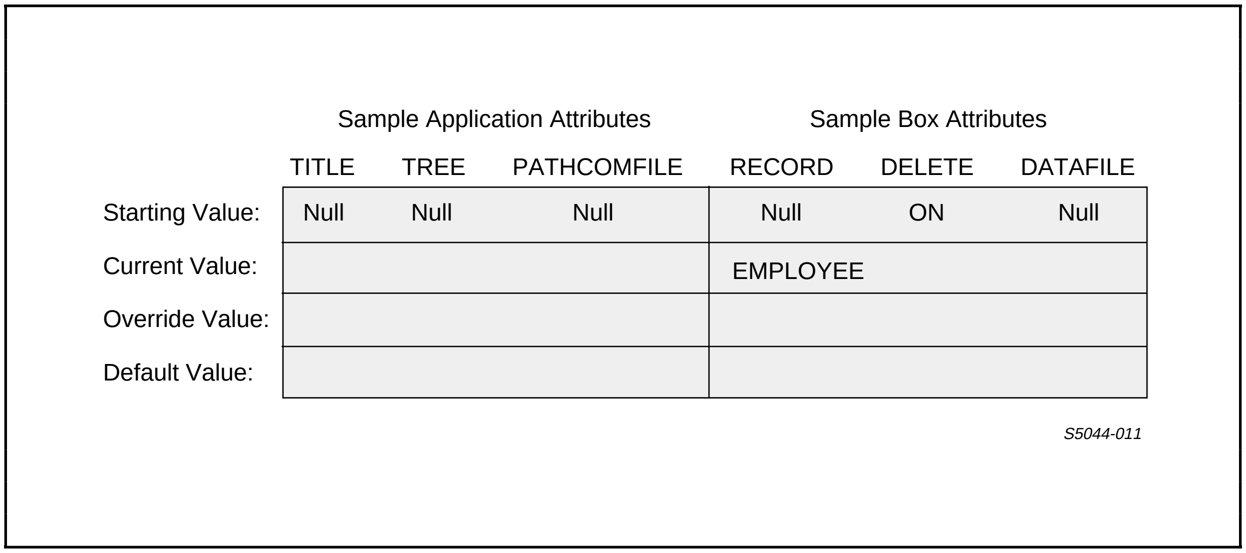


Figure 2-5. Adding a Box to the Object Table

When ENABLE adds "employ-box" to the object table, it removes any override or default values from the attribute table. Figure 2-6 shows the contents of the attribute table after ENABLE adds "employ-box."



	Sample Application Attributes			Sample Box Attributes		
	TITLE	TREE	PATHCOMFILE	RECORD	DELETE	DATAFILE
Starting Value:	Null	Null	Null	Null	ON	Null
Current Value:				EMPLOYEE		
Override Value:						
Default Value:						

S5044-011

Figure 2-6. Contents of Attribute Table After ADD BOX Command

Notice that ENABLE does not remove any current values from the attribute table. If you were to add another box without supplying either an override value or a new current value for the RECORD attribute, ENABLE would use "employee" as the value of the RECORD attribute for the new box.

Once ENABLE has added a box to the object table, you cannot change the description of the box. You can, however, remove the box from the object table with a DELETE command, supply a new description of the box, and use the ADD command to add the newly described box to the object table.

Describing and Adding an Application

The values of the application attributes describe an application. You can supply values for these attributes by using either of the following commands:

- The SET APPL command--which supplies a current value for an application attribute
- The ADD APPL command--which supplies an override value for an application attribute

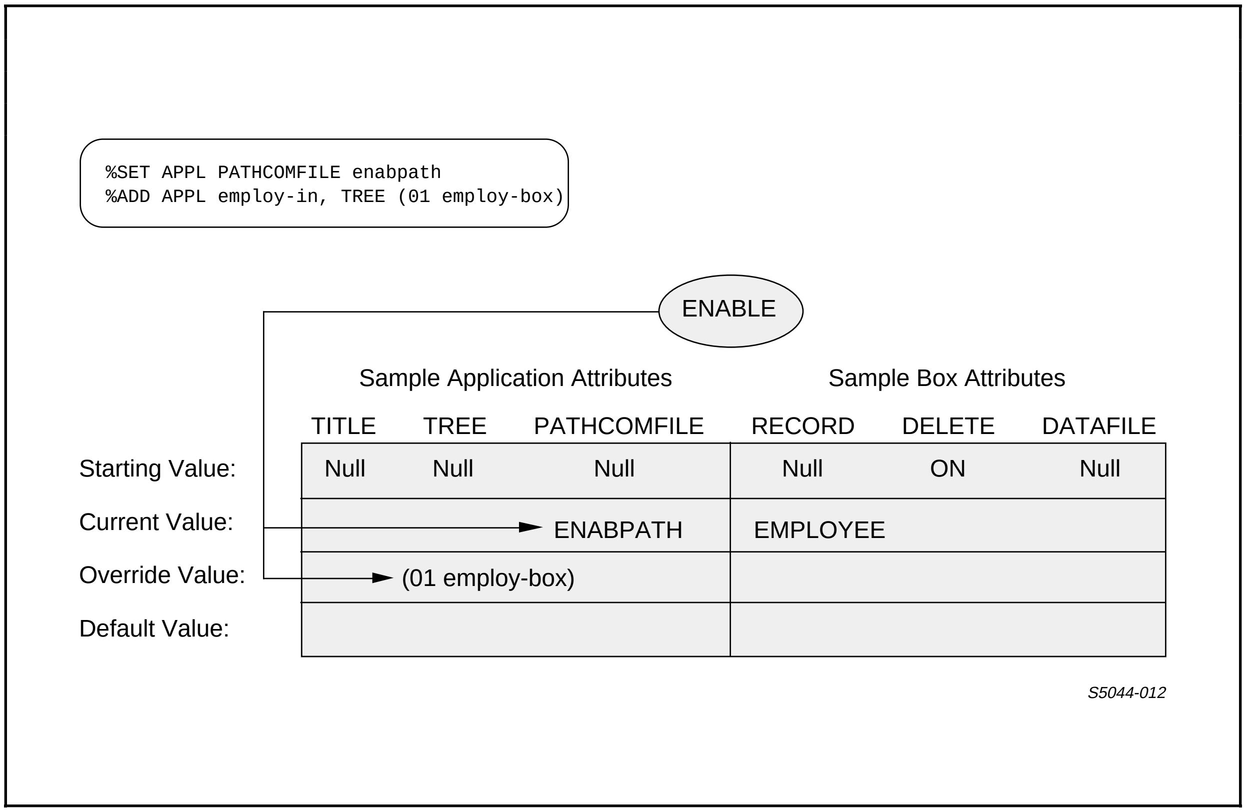
Suppose, for example, that you want ENABLE to write the PATHCOM command file for the "employ-in" application to a file named "enabpath." You could supply this file name as a value for the PATHCOMFILE attribute by entering the following command:

```
   SET APPL PATHCOMFILE enabpath
```

To tell ENABLE that the "employ-in" application uses "employ-box," you can supply an override value for the TREE attribute by entering command:

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

Figure 2-7 illustrates the action that ENABLE takes when you enter the preceding SET and ADD commands.



```
%SET APPL PATHCOMFILE enabpath
%ADD APPL employ-in, TREE (01 employ-box)
```

	TITLE	TREE	PATHCOMFILE	RECORD	DELETE	DATAFILE
Starting Value:	Null	Null	Null	Null	ON	Null
Current Value:			ENABPATH	EMPLOYEE		
Override Value:		(01 employ-box)				
Default Value:						

Figure 2-7. Attribute Table With a Current and an Override Application-Attribute Value

Before ENABLE adds the "employ-in" application to the object table, it examines the contents of the attribute table to determine if it must supply any default values for application attributes. Notice that in Figure 2-7, the starting value of the TITLE attribute is null. Since the attribute table contains neither an override nor a current value, ENABLE supplies the application name as a default value.

Figure 2-8 illustrates the partial contents of the attribute table when ENABLE supplies a default value for the TITLE attribute.

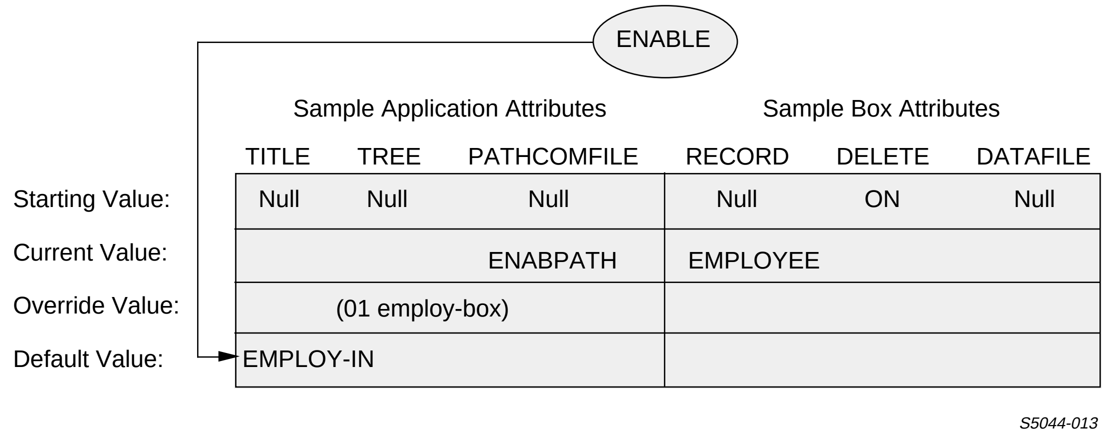


Figure 2-8. Attribute Table With a Default Application-Attribute Value

After ENABLE supplies the appropriate default values, it uses the contents of the attribute table to obtain the description of the "employ-in" application and adds this application to the object table, as illustrated by Figure 2-9.

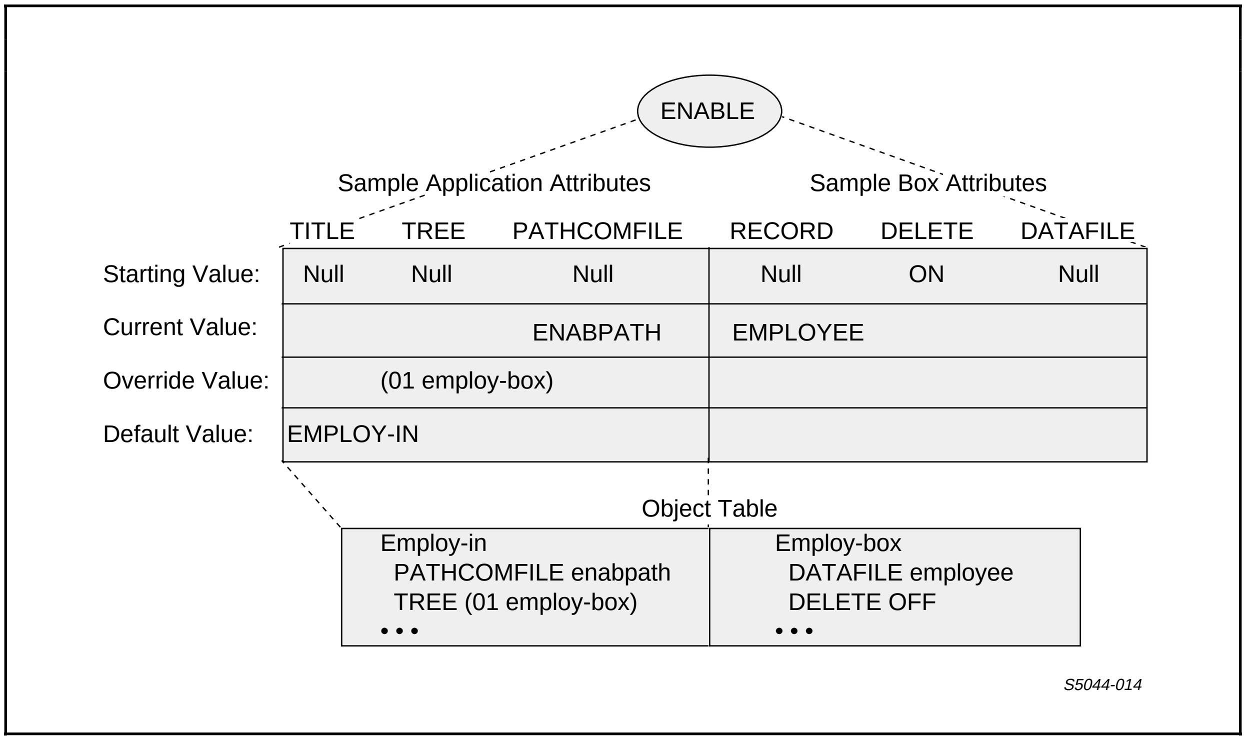
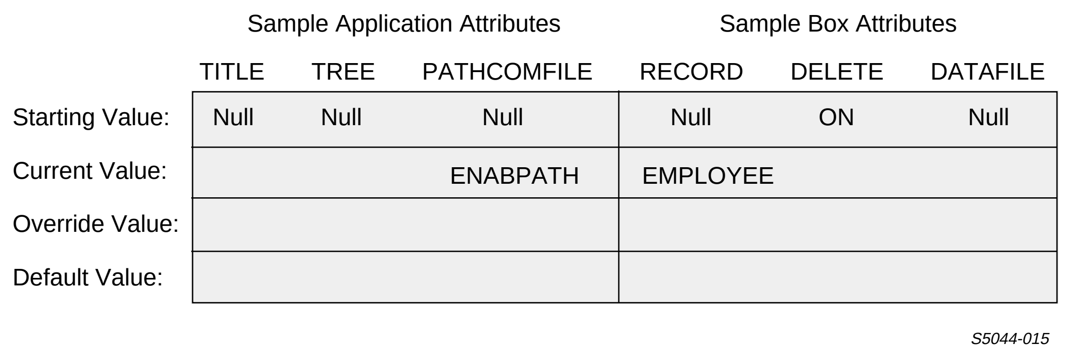


Figure 2-9. Adding an Application to the Object Table

After ENABLE adds "employ-in" to the object table, it removes any override and default values from the attribute table. Figure 2-10 illustrates the contents of the attribute table after ENABLE adds "employ-in."



	Sample Application Attributes			Sample Box Attributes		
	TITLE	TREE	PATHCOMFILE	RECORD	DELETE	DATAFILE
Starting Value:	Null	Null	Null	Null	ON	Null
Current Value:			ENABPATH	EMPLOYEE		
Override Value:						
Default Value:						

S5044-015

Figure 2-10. Contents Attribute Table After an ADD APPL Command

Notice that ENABLE does not remove the current value of any attribute. If, at this point, you added another application without supplying any application attribute values, ENABLE would try to write the PATHCOM command file for this application to "enabpath," the file identified by the current value of the PATHCOMFILE attribute.

Once ENABLE has added an application to the object table, you cannot change its description by supplying attribute values. You can, however, use the DELETE APPL command to remove the application from the object table, supply a new description, and use the ADD command to add the newly described application to the object table.

Generating an Application

To indicate that you want to generate an application, use the GENERATE command. The following command generates the "employ-in" application:

```
   GENERATE APPL employ-in
```

When you enter the GENERATE command, ENABLE uses the contents of the object table to generate the application, as illustrated in Figure 2-11.

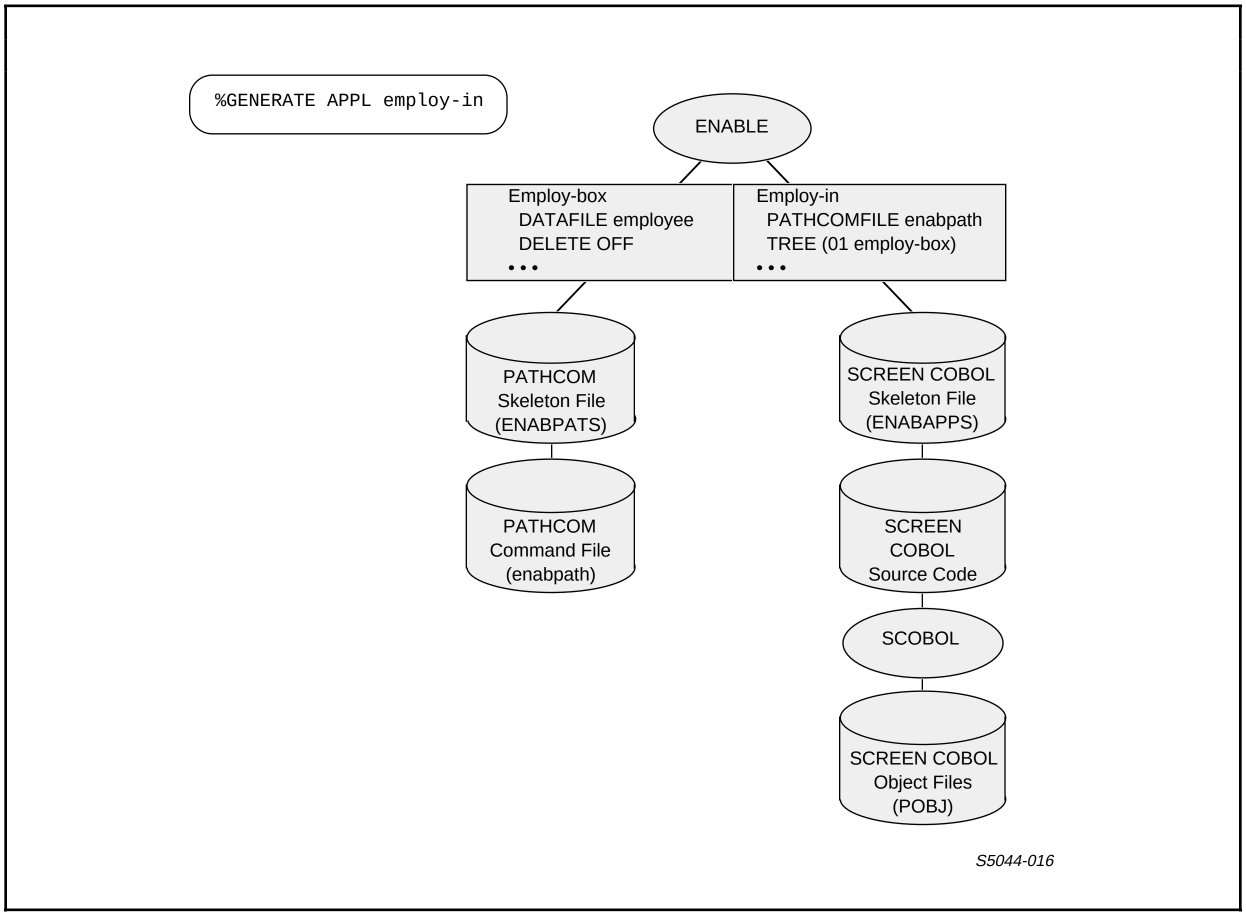


Figure 2-11. Generating an Application

To maintain compatibility with a previous version, ENABLE provides many default values. You can use these defaults to generate an application that accesses a single data base file. For example, you can generate the application described in the preceding paragraphs by entering the following ENABLE command:

```
   GENERATE employ-in, RECORD employee, PATHCOMFILE enabpath
```

Section 4 describes the recommended method of generating single-file applications; Section 5 describes the recommended method of generating multifile applications.

SUPPLYING A DATA BASE

Before you can execute an application generated by ENABLE, you must supply a data base consisting of one or more files, as described below. Refer to Section 3 for information about developing, describing, and creating a data base for your application.

What is a Data Base?

A data base is an organized collection of information, usually stored on a computer system. This information is represented by data in a standard format for ease of access by one, or by several application programs. A data base often serves as a repository for the information needed to perform certain functions in a commercial, scientific, or business enterprise.

Broadly speaking, the format of data in a data base is composed of three elements. Fields form the smallest units, describing one feature of interest. A record is a set of fields used to describe one complete item. A file is a set of records that describe similar items.

Fields

The smallest named unit of data in a data base is a field. Each field has a name and occupies a specific location in relation to other fields.

Fields can contain data that belongs to one of three broad categories: alphanumeric, alphabetic, or numeric. Alphanumeric fields contain data composed of letters of the alphabet, spaces, digits, and special symbols like the hyphen. Alphabetic fields contain data composed of letters of the alphabet and space characters. Numeric fields contain digits, minus or plus signs, and decimal points.

A group field consists of two or more continguous fields. Each field within a group field has its own field name. In addition, the group itself has a name that can be used to refer to all the fields. As an example, consider a group named "address." This group consists of fields named "street," "city," "state," and "zip-code."

The characters that are stored in a field are called the field value. When more than one value is associated with a field, the field is said to contain repeating field values. The field itself is called a repeating field or group.

Records

A record is a collection of associated fields. Figure 2-12 shows a typical employee record.

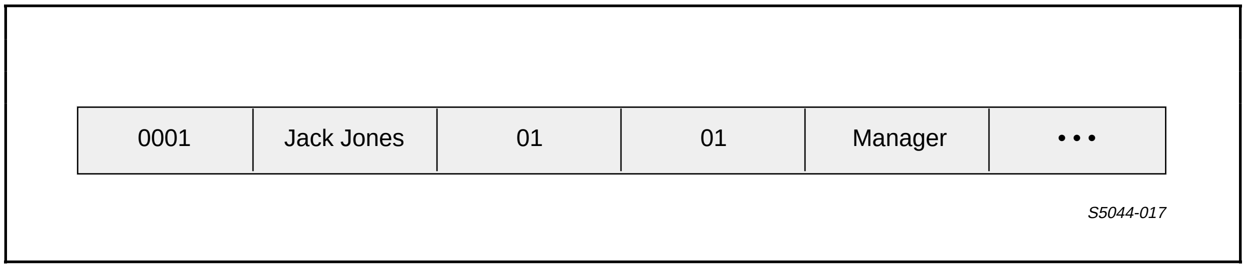
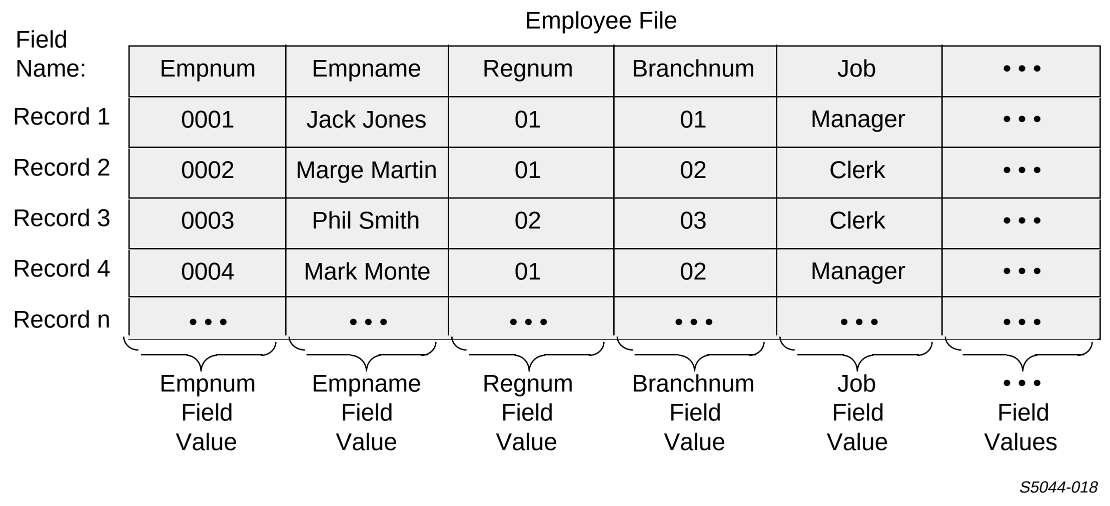


Figure 2-12. Sample Employee Record

An employee record contains the field values that concern a specific employee.

Files

A file is a collection of records. A file has a name but does not occupy a specific location in relation to other files. A file that contains a collection of employee records could be named "employee." Figure 2-13 shows an example of such a file.



Employee File

Field Name:	Empnum	Empname	Regnum	Branchnum	Job	• • •
Record 1	0001	Jack Jones	01	01	Manager	• • •
Record 2	0002	Marge Martin	01	02	Clerk	• • •
Record 3	0003	Phil Smith	02	03	Clerk	• • •
Record 4	0004	Mark Monte	01	02	Manager	• • •
Record n	• • •	• • •	• • •	• • •	• • •	• • •
	Empnum Field Value	Empname Field Value	Regnum Field Value	Branchnum Field Value	Job Field Value	• • • Field Values

S5044-018

Figure 2-13. Sample Employee File

Notice that this file resembles a table. Tandem computer systems store information in relational data base form. With a relational data base, you can regard each file as a two-dimensional table, where the columns correspond to fields, and the rows correspond to records.

Key Fields

A key field identifies a specific record within a file. Sometimes more than one field is needed to identify a record. When two or more contiguous fields are used to identify a specific record, the combined fields are called a composite key field.

A primary key field uniquely identifies a particular record. Only one primary key field can exist for a record. A likely primary key field for an employee record is empnum because only one employee can have this unique number.

An alternate key field identifies records with a common property. A record can have more than one alternate key. When a record has an alternate key, the application can use alternate search paths to retrieve data.

ESTABLISHING A PATHWAY SYSTEM

An ENABLE application must exist within a PATHWAY system. PATHWAY, a part of the ENCOMPASS distributed data base management system, supplies programs, procedures, and structures that define and control requester and server processes (programs). PATHWAY also provides a program that handles the terminal for your application.

ENABLE supplies you with many of the components necessary to establish a PATHWAY system. To establish such a system and execute an ENABLE application, you must:

1. Create a PATHWAY monitor process (PATHMON). PATHMON controls the activity of processes and devices within the PATHWAY system.

2. Create a PATHCOM process (PATHCOM). PATHCOM acts as the command interface to PATHMON.

3. Submit the PATHCOM command file, produced by ENABLE, to PATHCOM, which then passes the information on to PATHMON.

PATHMON uses the information in the PATHCOM command file to:

- Define a Terminal Control Process (TCP)--The TCP controls the screen and sends instructions to the General Server.

- Define the requester program generated by ENABLE--When executed, the requester process accepts requests from the screen and sends messages to the General Server.

- Define the General Server program supplied by ENABLE--The General Server opens the data base file and implements the request.

Figure 2-14, which is divided into two parts, illustrates the events that occur when you (1) establish a PATHWAY system and (2) execute an application.

Sections 4 and 5 provide more information about establishing a PATHWAY system and executing an application.

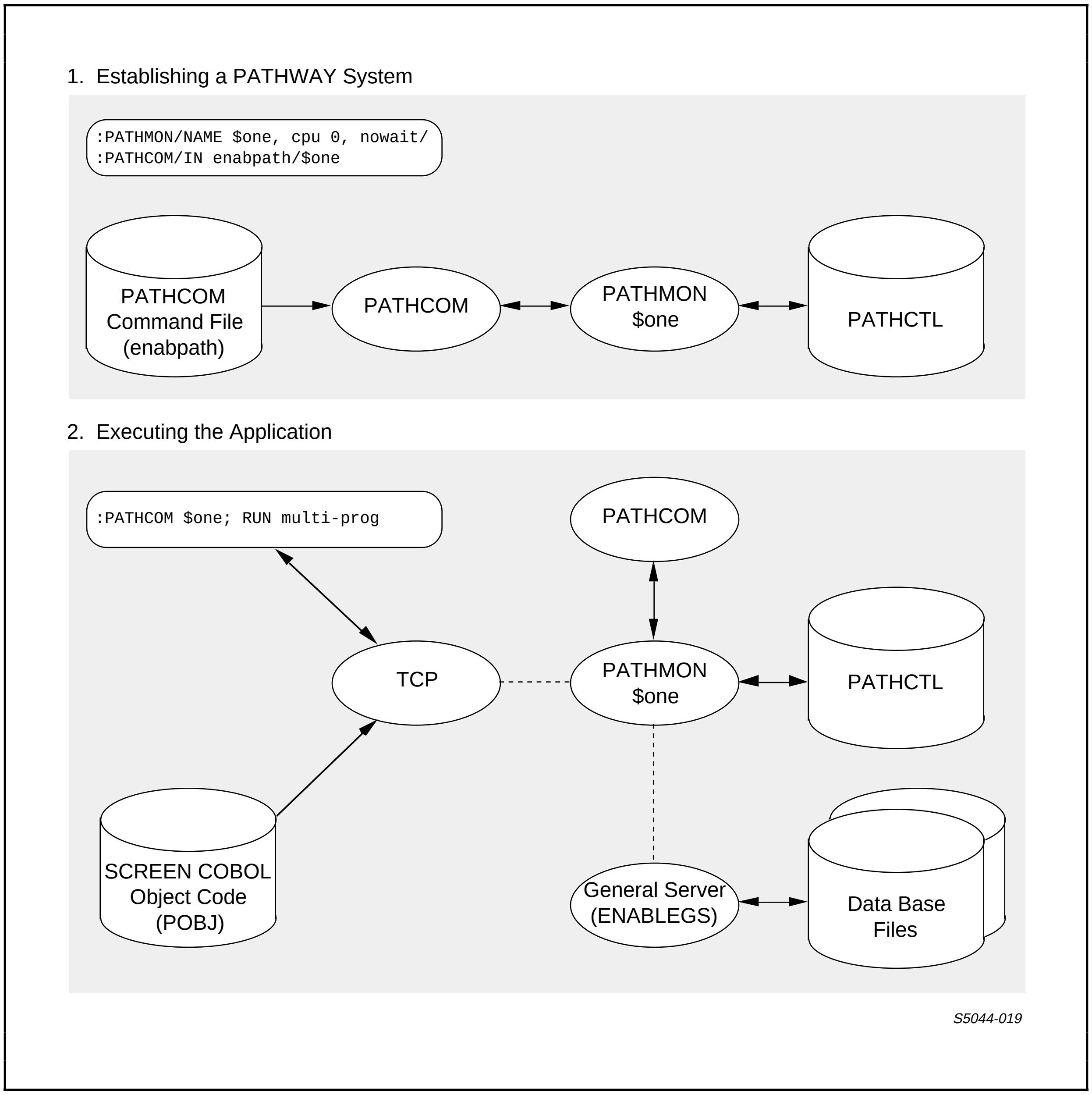

1. Establishing a PATHWAY System
:PATHMON/NAME $one, cpu 0, nowait/
:PATHCOM/IN enabpath/$one
PATHCOM Command File (enabpath)
PATHCOM
PATHMON $one
PATHCTL
2. Executing the Application
:PATHCOM $one; RUN multi-prog
PATHCOM
TCP
PATHMON $one
PATHCTL
SCREEN COBOL Object Code (POBJ)
General Server (ENABLEGS)
Data Base Files
S5044-019

Figure 2-14. Establishing a PATHWAY System and Executing an Application

SECTION 3

DEVELOPING, DESCRIBING, AND CREATING A DATA BASE

If the information your department uses is already stored in an existing data base, you can use that data base when you generate your ENABLE applications. If such a data base does not already exist, you must describe and create one before using ENABLE to generate an application to maintain it.

The discussion below follows the step-by-step process used to describe and create the data base files used for many of the examples in this guide. This sample data base comprises two files that store information used by a personnel department.

DEVELOPING A DATA BASE

Since many users of ENABLE also use ENFORM to obtain reports, the approach described in the following paragraphs results in a data base that can be used by ENFORM as well as ENABLE.

Briefly, the tasks involved in developing the sample personnel data base are:

1. Identifying the classes of information in which the personnel department is interested. (Each class of information will be a data base file.)

2. Listing the characteristics (or items) of data associated with each class of information. (Each data item will be a field within its associated file.)

3. Drawing a diagram of the relationships between the classes of data

4. Listing the fields within each file

5. Selecting an appropriate file type for each file

6. Choosing key fields for each file

Identifying the Classes of Information

You can determine the appropriate classes of data for the personnel department by considering the present flow of information; for example:

1. When an employee is hired, the clerk performing the interview fills out two forms:

   --The first form lists the employee's name, age, and sex. The clerk completes this form with information (such as the employee's salary) supplied by the employee's department manager.

   --The second form contains general information about the employee's dependents. On this form, the clerk lists each dependent's name, age, and sex.

2. Once a month, someone from the department updates a list that provides vacation information. This list identifies the number of vacation days available for each employee.

3. If an employee receives a raise, a clerk retrieves the employee's file and changes the salary information.

4. Whenever an employee informs the personnel department of a new dependent, someone in the department must add this information to the appropriate form.

After reviewing the current flow of information, you might conclude that the personnel department is interested in the classes of information shown in Figure 3-1.

EMPLOYEE	Each employee for which the personnel department is responsible.
DEPENDENTS	The dependents of these employees.

Figure 3-1. Classes of Information

Listing the Data Items

A study of the information flow yields the list of data items shown in Figure 3-2.

EMPLOYEE	
empnum	A unique identifying number
empname	The name of the employee
regnum	A number that identifies the region in which the employee works
branchnum	A number that identifies the branch for which the employee works
job	The job title of the employee
age	The employee's age
salary	The employee's current monthly salary
vacation	The number of vacation days accrued by the employee
DEPENDENTS	
emp-no	The identification number of an employee
empname	The name of the employee
branchnum	The branch number of the employee
regnum	The region number of the employee
dependent-name ...	The name of a dependent
relationship ...	The relationship between the dependent and the employee
dependent-age ...	The dependent's age

The symbol (...) indicates that a data item may be repeated.

Figure 3-2. List of Data Items

Drawing the Relationship Between Classes of Information

A drawing of the relationships that exist between classes of information often helps you discover problems in these relationships. Such a drawing is also helpful later when you use ENABLE to generate an application that accesses the information in these classes.

Since only two classes of data exist for the personnel data base, you can determine the relationship between these classes of data by drawing a diagram similar to the one shown in Figure 3-3.

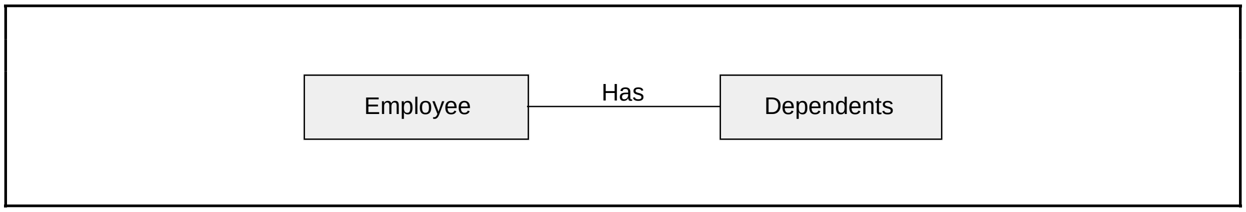


Figure 3-3. Relationship Between Classes of Information

Once you have drawn this diagram, the next step is to examine the relationship to determine how many occurrences of one class relate to occurrences of the other. Several different types of relationships can exist:

• one-to-one	For each occurrence in one class of information, there is exactly one occurrence in the other class.
• one-to-many	For each occurrence in one class of information, there are zero or more occurrences in the other class.
• many-to-many	Zero to many occurrences in one class of information can be associated with zero to many occurrences in the other class.

Consider the relationship between employee and dependents. For each employee occurrence, there is one dependent occurrence. Figure 3-4 illustrates this relationship.

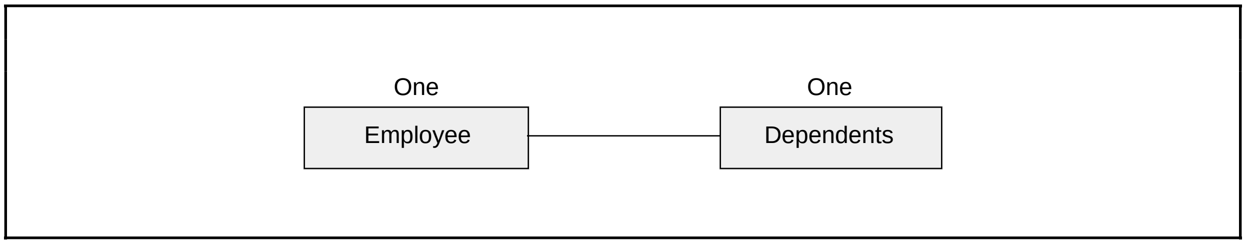


Figure 3-4. One-to-One Relationship

After you determine the type of relationship that exists between the classes of information, ask yourself the following questions:

- Are any of the data items associated with more than one class of information? If so, is this duplication necessary?

- Are any of the data items repeated in the same class of data?

Depending on the answers to these questions, you might need to draw a new diagram of the relationships between your classes of information.

Are Data Items Associated With More Than One Class?

Some employee data (an employee number, name, branch number, and region number) appears with both employee and dependents. To eliminate some of this redundant information, you could remove the employee name, branch number, and region number from the dependents information. The employee number should remain with both, however, so that you can form a logical connection between these classes when you use either ENABLE or ENFORM.

Are Data Items Repeated in the Same Class of Data?

Several data items (dependent-name, relationship, and dependent-age) can be repeated. Figure 3-5 shows some sample data from this class of information.

Sample Data for Dependents

empnum	dependent-name	relationship	dependent-age
0001	Jane Jones Bill Jones Mark Jones	S C C	33 12 10
0002	Judy Martin	C	17
0004	Leslie Monte David Monte Sue Monte	S C C	27 3 2
•••	•••	•••	•••

Figure 3-5. Dependents Class with Repeating Data

You can remove this repeating data from the dependents class by using a technique called "normalization." To normalize the dependents data, you could include "empnum" with each occurrence of "dependent-name," "relationship," and "dependent-age." Figure 3-6 illustrates how the sample dependent data appears when you eliminate repeating data.

Sample Data for Dependents

empnum	dependent-name	relationship	dependent-age
0001	Jane Jones	S	33
0001	Bill Jones	C	12
0001	Mark Jones	C	10
0002	Judy Martin	C	17
0004	Leslie Monte	S	27
0004	David Monte	C	3
0004	Sue Monte	C	2
•••	•••	•••	•••

Figure 3-6. Dependents Class with Repeating Data Removed

Although normalization requires you to include the same data item ("empnum") with several occurrences of the dependent data, this process simplifies your data base files and results in a data base that can be handled by both ENABLE and ENFORM.

Drawing the New Relationship

After you remove any data items that are associated with more than one class of data and eliminate repeating data from each class, you can draw a new diagram that shows the relationship between the classes. Since the repeating information has been removed from the dependents class, each employee occurrence is now related to several dependents occurrences. Figure 3-7 illustrates this new relationship.

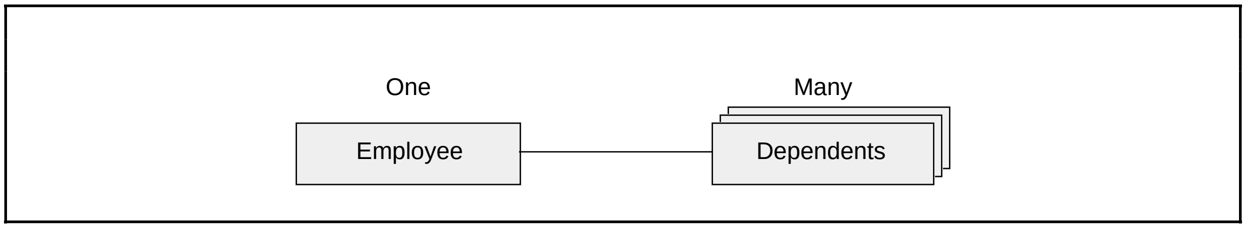


Figure 3-7. One-to-Many Relationship

Listing the Fields in Each File

At this point, you have identified the two files in the personnel data base: employee and dependents. Since dependents is too long to be a valid file name (eight character maximum length), you might shorten this name to "depend." The next step is to list the fields in each file as shown in Figure 3-8.

EMPLOYEE	empnum, empname, regnum, branchnum, job, age, salary, vacation
DEPEND	emp-no, dependent-name, relationship, dependent-age

Figure 3-8. List of Fields in Both Files

Notice that both files contain a field that represents an employee identification number. The presence of such duplicate fields is essential because you need these fields to provide logical connections when you use ENABLE to generate an application or use ENFORM to request a report.

After you list the fields in each file, you are ready to select the physical structure for the file.

Selecting the Appropriate File Type

You can select one of the following file types for your data base files:

- Key-sequenced--In a key-sequenced file, each record has a primary key and up to 255 alternate keys. If you select a key-sequenced file type, you can delete, insert, read, and update records in the file. To select a record you want to read, you can use either the primary or alternate key.

- Relative--In a relative file, each record has a unique record number and can have up to 255 alternate keys. The record number is a unique value that corresponds to the physical position of a record within the file. If you select a relative file type, you can delete, insert, read, or update records within the file. To select a record you want to read, you can either use the record number as a primary key or use an alternate key.

- Entry-sequenced--In an entry-sequenced file, each record has a unique record number and up to 255 alternate keys. The record number corresponds to the order in which a record is stored in the file. If you select an entry-sequenced file type, you can insert, read, or update records in the file; you cannot, however, delete records from the file. To select a record you want to read, you can either use the record number as a primary key or use an alternate key.

- Unstructured--In an unstructured file, each record has a unique record number that serves as a primary key. Alternate keys cannot exist. If you select an unstructured file type, you can insert, read, or update records in the file; you cannot, however, delete records from the file. To select a record you want to read, you must use the record number as a primary key.

Generally, you should select a key-sequenced file type for your data base files. This file type supplies you with the most flexibility of use. Of the other file types, you might want to select:

- A relative file type if you have a use for the record number field

- An entry-sequenced file type if you plan always to enter data in your file sequentially

A key-sequenced file type is the best choice for both files in the personnel data base. A relative file type is not necessary because no logical use for the record number field exists. An entry-sequenced file type is not satisfactory because the personnel department enters data in both files randomly and sequentially. An unstructured file type is unsatisfactory because, with such a file type, the personnel department could not use alternate keys to find an "employee" or "depend" record.

After you select a file type for a file, you are ready to choose key fields for that file.

Choosing Key Fields

Assuming that you have selected a key-sequenced file type for your data base files, you must now choose one primary key field for each. In addition, you can choose up to 255 alternate key fields for each file.

For the "employee" file, a good choice for a primary key is "empnum." One of the requirements for a primary key field is that it stores unique values. Since each employee has a unique employee number, "empnum" satisfies this requirement. Good candidates for alternate keys in this file are "empname," "regnum," and "branchnum." Instead of making both "regnum" and "branchnum" separate alternate key fields, you could identify a group field ("dept") of which these fields are a part and make that group an alternate key field. When you choose a group as a key field, that group is said to be a composite key field.

For the "depend" file, neither "emp-no" nor "dependent-name" uniquely identify a record. These fields are not unique because the same employee number can appear with several dependents, and several dependents could have the same name. If you provide a number that identifies each dependent of a given employee, you could identify a group ("dep-key") consisting of "emp-no" and the dependent number ("dependent-no"). You could then use this group as a composite primary-key field. Although "dependent-name" is not unique, and therefore cannot serve as a primary key, it is a good choice for an alternate key field.

When you are choosing key fields for your files, consider making any field (such as "empnum") that also appears in another file a key field of the other file. If you do this, you can use ENABLE to generate an application that can access both files.

After you choose the key fields for your files, you are ready to use the Data Definition Language (DDL) to describe your data base.

DESCRIBING A DATA BASE

Before you can use ENABLE to generate an application, you must use DDL to describe each of the files in your data base. When you use DDL, you create a data dictionary.

A data dictionary is a set of files that documents the structure of the data base. The dictionary documents the structure of each file with a record description that you create. When you create a record description, you can provide the following information:

- Field names
- The type of data (either alphanumeric, alphabetic, or numeric) that a field represents
- The size of a field
- A label for the field that can be used by both ENABLE and ENFORM
- The name of a file whose organization is described by the record description
- The structure (key-sequenced, relative, entry-sequenced, or unstructured) of a file that the record description describes
- Key fields

When you use DDL to create the record description and the dictionary, you can also produce File Utility Program (FUP) source code with which you can create the data base file.

Using DDL to Create a Record Description

To create a record description, you use the DDL RECORD statement. Although you can use DDL interactively, the recommended procedure is to enter the DDL source code (in this case, the DDL RECORD statement) in an edit-type file.

Consider Figure 3-9, which shows the DDL RECORD statements of the two files in the sample data base. The numbers that appear to the right of this figure refer to notes that describe different portions of the RECORD statements.

```
RECORD employee.                                              (1)
 FILE IS employee     KEY-SEQUENCED.                          (2)
 02 empnum          PIC 9(4)    HEADING "Employee Number".  (3)
 02 empname         PIC X(18)   HEADING "Employee Name".    (4)
 02 dept                        HEADING "Department".       (5)
   04 regnum        PIC 9(4)    HEADING "Region Number".    (6)
   04 branchnum     PIC 9(4)    HEADING "Branch Number".
 02 job             PIC X(12)   HEADING "Job Title".
 02 age             PIC 9(2)    HEADING "Age".
 02 salary          PIC 9(4)V99 HEADING "Salary".           (7)
 02 vacation        PIC S99     HEADING "Vacation".         (8)
 KEY IS empnum.                                             (9)
 KEY "en" IS empname.                                       (10)
 KEY "dp" IS dept.                                          (11)
 END                                                        (12)

 RECORD dependents.                                         (13)
 FILE IS depend       KEY-SEQUENCED.                        (14)
 02 dep-key.                                                (15)
   04 emp-no          PIC 9(4)  HEADING "Employee Number".
   04 dependent-no    PIC 9(2)  HEADING "Dependent Number".
 02 dependent-name    PIC X(18) HEADING "Dependent Name.".
 02 relationship      PIC A.                                (16)
 02 dependent-age     PIC 99.
 KEY 0 IS dep-key.                                          (17)
 KEY "dn" IS dependent-name.                                (18)
 END
```

NOTES

(1) Identifies the subsequent information as a RECORD statement and supplies the record description name "employee."

Figure 3-9. Sample DDL RECORD Statements (Continued next page)

(2) Identifies a file name ("employee"); also indicates the file type (KEY-SEQUENCED).

(3) Names a field, "empnum." The PIC clause identifies the size of this elementary field (four characters) and indicates its data type (numeric). Note the two digits to the left of the field name. These digits are called level numbers. All fields have level numbers to indicate their relationship to other fields. You can select any digits from 1 to 50 for level numbers.

Notice the HEADING clause that appears with this and many other fields. Including such a clause provides you with a method of supplying your own screen labels when you use ENABLE to generate an application.

(4) Names a field, "empname." The PIC clause identifies the size of this elementary field (18 characters) and indicates its data type (alphanumeric).

(5) Names a field, "dept." Notice that this field does not have a PIC clause. The absence of a PIC clause indicates that the field is a group field. A group field consists of all of the immediately following elementary fields with lower level numbers. Be aware of the way that DDL evaluates level numbers. For DDL, a level number of 02 is higher than a level number of 03.

(6) Names a field, "regnum." The PIC clause identifies the size (four characters) of this elementary field and indicates its data type (numeric). Note that the level number (04) associated with this field is lower than the level number (02) associated with the group field ("dept") of which "regnum" is a part.

(7) Names a field, "salary." The PIC clause identifies the size (six characters) of this elementary field and indicates its data type (numeric). The symbol V in the PIC clause indicates an implied decimal point. The decimal point is not actually stored in the file.

Figure 3-9. Sample DDL RECORD Statements (Continued)

(8) Names a field, "vacation." The PIC clause identifies the size (three characters) of this elementary field and indicates its data type (numeric). The symbol "S" in the PIC field indicates that the value for the field can include a plus or minus sign.

(9) Identifies a key field, "empnum." The absence of a key specifier in this KEY clause indicates that "empnum" is the primary key field. Since the "employee" file is identified as a key-sequenced file, the RECORD statement must identify a primary key field.

(10) Identifies a key field, "empname." The presence of a two character mnemonic (en) in the KEY clause indicates that "empname" is an alternate key field.

(11) Identifies a composite key field, "dept." A composite key field is a key that consists of a group field. The presence of the two-character mnemonic (dp) indicates that "dept" is an alternate key field.

(12) Terminates the RECORD statement.

(13) Identifies the beginning of a new RECORD statement and supplies the record description name "dependents."

(14) Supplies a file name ("depend") and identifies the file type (KEY-SEQUENCED).

(15) Defines a group field ("dep-key").

(16) Defines a field named "relationship." The PIC clause associated with this field indicates that its data type is alphabetic and its size is one character.

(17) Identifies "dep-key" as the primary key. Note that since "dep-key" is a group field, this primary key is a composite key.

(18) Identifies "dependent-name" as an alternate key field.

Figure 3-9. Sample DDL RECORD Statements (Continued)

ENABLE Limitations on DDL

Because ENABLE uses record descriptions to generate SCREEN COBOL source code, the syntax of a RECORD statement must be compatible with SCREEN COBOL representation capabilities. Specifically, this affects the syntax that you can use for the RECORD statement as follows:

- If you use a TYPE clause or a user-defined TYPE (see the Data Definition Language (DDL) Reference Manual for details), the data type must be either CHARACTER or BINARY.
- BINARY 8, FLOAT 32, FLOAT 64, COMPLEX, LOGICAL 2 and LOGICAL 4 data items are not supported by ENABLE.
- When you define numeric fields, the PIC clause cannot define more than 18 digits; for example, both PIC 9999999999999999999 and PIC 9(19) are illegal.
- If you use an OCCURS clause, the OCCURS clause nesting must not exceed four levels. The number of occurrences must not exceed 999.
- When you specify elementary fields, the size of each field must not exceed 256 bytes.
- If you add the values specified for the size descriptions of all the fields in the record description, the total number of bytes must not exceed 2046.
- You can use a FILLER clause; however, the application will not display the filler on the screen.
- You can use a REDEFINES clause; however, ENABLE ignores the redefined structure.
- You can use a RENAMES clause; however, ENABLE ignores the renamed structure.
- If you identify the entire record as a key field, an ENABLE application will use the first group or elementary item in the record as the key.
- If you use an OCCURS DEPENDING ON clause, ENABLE will use the maximum number of occurs items. Refer to the ENABLE Reference Manual for more information.

Refer to the Data Definition Language (DDL) Reference Manual for more information about the RECORD statement.

Creating a Dictionary

Once you write your RECORD statements in an edit-type file, you can use the DDL compiler either to create a new dictionary for the record descriptions or to add the record descriptions to an existing dictionary. By including a special command when you use the DDL compiler, you can also tell the compiler to write FUP file-creation commands that you can use later to create your data base files.

To use the DDL compiler, enter a command with the following DDL COMMAND syntax from the command interpreter:

```
DDL/IN <ddl-source-file>/ [ <command> ] [ , <command> ... ]
```

<ddl-source-file>

is the name of an edit-type file containing one or more DEFINITION or RECORD statements or DDL compiler commands.

<command>

is any DDL compiler command. Refer to the Data Definition Language Reference Manual for a complete list of DDL compiler commands.

DDL compiler commands that are relevant when creating a data base for an ENABLE application are as follows:

DICT [!]

creates a new dictionary for the record descriptions or adds the record descriptions to an existing dictionary. To purge an existing dictionary and add the record to a new dictionary, include the exclamation point (!).

FUP <file-name> [!]

tells the DDL compiler to create <file-name> if it doesn't already exist and to write the FUP file-creation commands to this file. To purge any previous contents of <file-name>, include the exclamation point (!).

If your RECORD statements for the sample data base are in an edit-type file named "ddlsrc1," you can create a dictionary for these record descriptions by entering the following:

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

This command tells the DDL compiler to:

1. Compile the source code on "ddlsrc1"

2. Open and update a data dictionary on the default volume and subvolume

3. Create and write FUP source commands in "fupsrc1"

After you compile your DDL source code, you can use the FUP source file produced by DDL to create your data base.

CREATING A DATA BASE

Before you execute an application, the data base file (or files) to be used by the application must exist. To create these files, use the FUP commands produced by the DDL compiler. This assures that the structure and organization of a file is the same as the structure and organization described by the corresponding record description.

This is particularly important if you use ENABLE to generate an application that access one of these files. An application generated by ENABLE retains information about a file, such as the file type, record organization, and key field location. When you execute the application, the General Server checks to see if a physical file matches the information retained by the application. If a file does not match, the application displays an error message stating:

```
   Regenerate program:  file has changed.
```

Figure 3-10 shows an annotated example of the FUP file-creation commands created by the DDL compiler for the sample data base files.

```
< SCHEMA PRODUCED DATE - TIME :  1/08/84 15:45:21       (1)
< SECTION EMPLOYEE
< Record EMPLOYEE created on 01/08/84 at 15:45
RESET
   SET ALTKEY ( "en," KEYOFF 4, KEYLEN 18, FILE 0 )    (2)
   SET ALTKEY ( "dp," KEYOFF 22, KEYLEN 8, FILE 0 )
   SET NO ALTCREATE
   SET ALTFILE ( 0, EMPLOYE0 )                         (3)
   SET TYPE K                                          (4)
   SET KEYOFF 0                                        (5)
   SET KEYLEN 4                                        (6)
   SET REC 54                                          (7)
   SET BLOCK 512                                       (8)
   SET IBLOCK 512                                      (9)
CREATE EMPLOYEE                                        (10)
   RESET                                               (11)
   SET TYPE K                                          (12)
   SET KEYLEN 24
   SET REC 24
   SET BLOCK 512
   SET IBLOCK 512
CREATE EMPLOYE0                                        (13)
< SECTION DEPENDENTS
< Record DEPENDENTS created on 01/08/84 at 15:45
RESET
   SET ALTKEY ( "dn," KEYOFF 6, KEYLEN 18, FILE 0)     (14)
   SET NOALTCREATE
   SET ALTFILE ( 0, DEPEND0)                           (15)
   SET TYPE K
   SET KEYOFF 0
   SET KEYLEN 6
   SET REC 29
   SET BLOCK 512
   SET IBLOCK 512
CREATE DEPEND                                          (16)
```

Figure 3-10. Sample FUP Source Commands (Continued next page)

```
   RESET
   SET TYPE K
   SET KEYLEN 26
   SET REC 26
   SET BLOCK 512
   SET IBLOCK 512
CREATE DEPEND0                                       (17)
```

NOTES

(1) Contains header information written by the DDL compiler

(2) Identifies an alternate key specifier ("en"), provides the location of this key (KEYOFF 4) in bytes relative to the start of a record, defines the key length (KEYLEN 18) in bytes, and identifies the alternate key file number (FILE 0). (FUP associates this alternate key with the actual alternate key file by means of the ALTFILE command described in note 3.)

(3) Identifies the name of the alternate key file ("employe0") associated with this data file

(4) Indicates that the data file is a key-sequenced (K) file

(5) Identifies the location (KEYOFF 0) in bytes of the primary key field

(6) Identifies the length (KEYLEN 4) in bytes of the primary key field

(7) Identifies the length (REC 54) in bytes of the records that can be stored by this data file

(8) Identifies the data block length in bytes (BLOCK 512)

Figure 3-10. Sample FUP Source Commands (Continued)

```
-----------------------------------------------------------------
|                                                               |
| (9)  Identifies the index block length in bytes (IBLOCK       |
|      512).  (Consider increasing this to 4096 bytes.)         |
|                                                               |
|(10)  Creates the named data file ("employee")                 |
|                                                               |
|(11)  Restores the file-creation command parameters to their   |
|      default settings                                         |
|                                                               |
|(12)  Indicates that the alternate key file is a               |
|      key-sequenced (K) file.  Alternate key files are always  |
|      key-sequenced files                                      |
|                                                               |
|(13)  Creates the alternate key file ("employe0")              |
|                                                               |
|(14)  Identifies an alternate key specifier ("dn"), provides   |
|      the location of this key (KEYOFF 6) in bytes relative    |
|      to the start of a record, defines the key length         |
|      (KEYLEN 18) in bytes, and identifies the alternate key   |
|      file number (FILE 0).  (FUP associates this alternate    |
|      key with the actual alternate key file by means of the   |
|      ALTFILE command described in note 15.)                   |
|                                                               |
|(15)  Identifies the alternate key file ("depend0")            |
|      associated with "depend"                                 |
|                                                               |
|(16)  Creates "depend"                                         |
|                                                               |
|(17)  Creates "depend0," the alternate key file for "depend"   |
|                                                               |
-----------------------------------------------------------------
```

Figure 3-10. Sample FUP Source Commands (Continued)

Determining File Size

When DDL writes FUP file-creation commands, it does not include a command to define the file size. The default file size is one page (4096 bytes) for the primary extent and one page for each secondary extent. Each file has 1 primary extent and up to 15 secondary extents that are allocated automatically by the computer system when they are needed.

These default extent sizes are small. Using them, a key-sequenced file with records 100 bytes long could store about 320 records. For most applications, you will want to make the extents larger.

When you determine the size of a file, estimate the number of records that the file will store. To do this you will have to find answers to such questions as, "How many people work for the company?" You might want to include space for records that are no longer active such as those that store information about people who have left the company. Consider how many records will be added to the file each month and leave some room for growth.

As a minimum, you will want to make the primary extent big enough to hold all of the records that are to be initially stored in the file. You can then let the system use the secondary extents for growth.

Once you have determined the number of records that your file will contain, add the following FUP statement to the FUP file-creation commands:

```
   SET EXT  (xxx RECS, yyy RECS)
```

where xxx is the number of records that are to fit in the primary extent and yyy is the number of records that are to fit in the secondary extent. Insert this command before the FUP CREATE commands that create the data file and the alternate key file. The FUP SET EXT command is the only way that you can define your file size.

When you include the SET EXT command, FUP uses the number of records you specify, the record size, and the size of the file block to calculate the number of pages required to hold your data.

Refer to the ENSCRIBE Programming Manual for more information about selecting an appropriate extent size.

Increasing Block Size

If you increase the size of the BLOCK parameter, you can enhance the efficiency of any application that accesses records from the file sequentially. Consider changing this parameter from BLOCK 512 to BLOCK 4096.

Using FUP to Create Your Files

To use FUP to create your files, enter the FUP command from the command interpreter using the following FUP command syntax:

```
-----------------------------------------------------------------
|                                                               |
|  FUP/IN <fup-source-file>/                                    |
|                                                               |
|  <fup-source-file>                                            |
|                                                               |
|    is the name of an edit-type file containing FUP commands.  |
|                                                               |
-----------------------------------------------------------------
```

Consider, for example, the following command:

```
   FUP/IN fupsrc1/
```

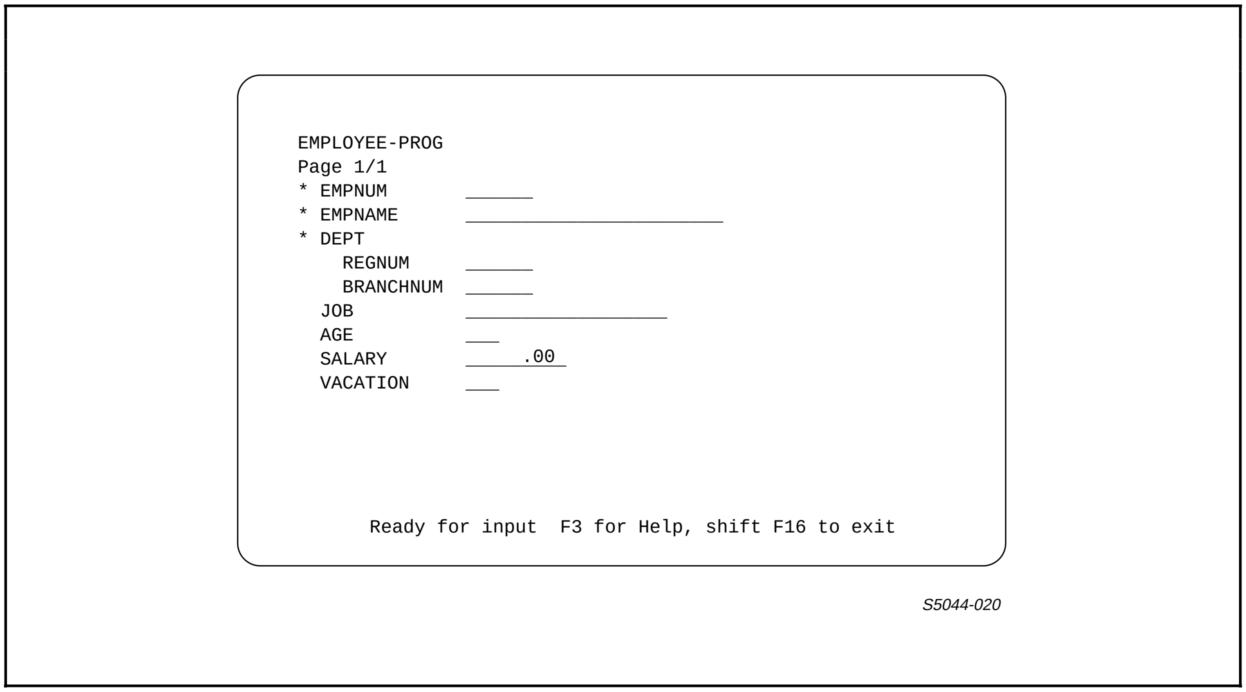
This command tells FUP to execute the commands in the edit-type file named "fupsrc1."

After you create the appropriate data base files, you are ready to use ENABLE to generate the application. Section 4 provides you with guidelines for generating and running an application that can access a single data base file. Section 5 provides guidelines for generating and running an application that can access two or more data base files.

SECTION 4

CREATING AND RUNNING A SINGLE-FILE APPLICATION

You can use ENABLE to generate an application through which you can enter or change data in a single data base file. When you execute the application, a screen similar to the one shown in Figure 4-1 appears.



```
EMPLOYEE-PROG
Page 1/1
* EMPNUM       ______
* EMPNAME      _______________________
* DEPT
    REGNUM     ______
    BRANCHNUM  ______
  JOB          __________________
  AGE          ___
  SALARY       _____.00_
  VACATION     ___

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

S5044-020

Figure 4-1. Screen Displayed by the Sample Employee-Prog Application

The application that displays this screen could be used to maintain employee information. To generate and execute this application, enter the series of commands shown in Figure 4-2. (Within the figure, arbitrary file names appear in lowercase letters. You can use these names exactly as shown or you can change them to avoid conflicts with existing file names.) Using these commands, you:

1. Create a data dictionary with the Data Definition Language (DDL).

2. Create a data base file with the File Utility Program (FUP).

3. Generate a single-file application with ENABLE.

4. Establish a PATHWAY system and run the application with the command interpreter, the Tandem text editor (EDIT), and PATHCOM.

After you generate and execute the application, you can enter data into the file and manipulate the data according to the instructions presented in Section 7.

```
STEP 1:  CREATE A DATA DICTIONARY

:EDIT ddlsrc1

*ADD
   1  RECORD EMPLOYEE.
   2  FILE IS employee KEY-SEQUENCED.
   3  02 EMPNUM      PIC 9(4)    HEADING "Employee Number".
   4  02 EMPNAME     PIC X(18)   HEADING "Employee Name".
   5  02 DEPT                    HEADING "Department".
   6    04 REGNUM    PIC 9(4)    HEADING "Region Number".
   7    04 BRANCHNUM PIC 9(4)    HEADING "Branch Number".
   8  02 JOB         PIC X(12)   HEADING "Job Title".
   9  02 AGE         PIC 9(2)    HEADING "Age".
  10  02 SALARY      PIC 9(4)V99 HEADING "Salary".
  11  02 VACATION    PIC S99     HEADING "Vacation".
  12  KEY IS EMPNUM.
  13  KEY "EN" IS EMPNAME.
  14  KEY "DP" IS DEPT.
  15  END
  16  //
*EXIT

:DDL/IN ddlsrc1/DICT, FUP fupsrc1 !

STEP 2:  CREATE A DATA BASE FILE

:FUP/IN fupsrc1/

STEP 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
```

Figure 4-2. Commands That Generate and Execute the Sample Employee-Prog Application (Continued next page)

```
STEP 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 $one,CPU 0, NOWAIT,OUT enablog/
   5  PATHCOM/IN singlpth/$one
   6  PATHCOM $one;RUN EMPLOYEE-PROG
   7  PATHCOM $one;shutdown,wait
   8  //

:OBEY enabex1
```

Figure 4-2. Commands That Generate and Execute the Sample Employee-Prog Application (Continued)

Section 3 describes how to create a dictionary and a data base file. The tasks that you perform to generate and execute your own single-file application are:

1. Generating the application

2. Establishing a PATHWAY system and executing the application

GENERATING A SINGLE-FILE APPLICATION

When you generate a basic single-file application, you use, with a few exceptions, either the starting or the default values of box or application attributes. (Refer to Section 2 for a list of these attribute values.) To generate such an application, perform the following tasks:

1. Start ENABLE by using the ENABLE run command in response to the command interpreter prompt, for example:

   ```
   :ENABLE
   ```

   ENABLE prompts for input by displaying the percent symbol (%). When this prompt appears, you can enter the ENABLE commands that describe your application.

2. Describe a box by supplying a value for the RECORD attribute. This attribute identifies the record description that ENABLE will use to obtain information about the data base file to be accessed. For the sample "employee-prog" application, supply "employee" as a record description name by using the following SET command:

   ```
   SET BOX RECORD employee
   ```

   When you supply a value for the RECORD attribute, ENABLE stores this value in the attribute table. ENABLE does not open a dictionary and obtain the record description until you add a box. If ENABLE cannot find a record description, it will issue an error message at that time.

3. Add a box that represents the data base file to be accessed by the application. When you add a box to the object table, you also provide a box name. For the sample "employee-prog" application, add a box named "employee" with the following ADD BOX command:

   ```
   ADD BOX employee
   ```

   ENABLE uses the contents of the attribute table to determine the description of the box. The value of the DATAFILE attribute identifies the data base file that a box represents. If you have not supplied a value for DATAFILE, ENABLE uses the file name contained in the record description as the default value of this attribute. For the sample "employee-prog" application, ENABLE uses "employee" as the file name.

4. Supply a value for the TREE attrtribute. When you add an application, ENABLE uses the value of the TREE attribute to associate a box with the application. For the sample "employee-prog" application, you associate a box named "employee" with an application by using the following SET command:

   ```
   SET APPL TREE (01 employee)
   ```

Notice the level number (01) that appears to the left of the box name. The value of this level number is significant only for multifile applications. (See Section 5 for more information.) For a single-file application, the only requirement for this level number is that it be any digit from 1 to 50.

5. Supply a value for the PATHCOMFILE attribute. This attribute identifies the file to which ENABLE is to write the PATHCOM commands. (After you generate your application, you use this PATHCOM command file to establish a PATHWAY system with which you can execute your application.) If you do not supply a value for this attribute, ENABLE does not write a PATHCOM command file. For the sample "employee-prog" application, you use the following SET command to identify "singlpth" as the name of the file for the PATHCOM commands:

   ```
   SET APPL PATHCOMFILE singlpth
   ```

   If you want ENABLE to write the PATHCOM commands to an existing file, include an exclamation point (!) with the file name. When you include this symbol, ENABLE replaces the contents of the file with the PATHCOM commands. If you tell ENABLE to write these commands to an existing file, that file must be an edit-type file.

6. Add the application to the object table. When you add an application, you also provide an application name. To add the sample "employee-prog" applicaton, use the following ADD command:

   ```
   ADD APPL employee-prog
   ```

   When you enter an ADD APPL command, ENABLE uses the contents of the attribute table to determine the description of the application. If you have not provided a value for the TITLE attribute (the value of this attribute identifies the screen title), ENABLE will use the application name as the default value and, hence, as the default screen title.

   After you add an application to the object table, you can either repeat steps 1 through 6 to describe and add more applications or generate the application you just added.

7. Generate the application by using the following command:

   ```
   GENERATE employee-prog
   ```

When you enter a GENERATE command, ENABLE displays the following messages:

```
Generating application:  application-name Please wait ...

Program generated:      application-name
```

8. Exit from ENABLE by entering the following command:

   ```
   EXIT
   ```

   Alternatively, you could exit from ENABLE by pressing the CTRL and Y terminal keys simultaneously.

When you complete these tasks, you are ready to establish a PATHWAY system to execute your application.

ESTABLISHING A PATHWAY SYSTEM AND EXECUTING AN APPLICATION

When you supply a value for the PATHCOMFILE attribute and generate an application, ENABLE writes PATHCOM commands to the named edit-type file. ENABLE includes all the commands necessary to configure (set up) a PATHWAY system. Figure 4-3 shows the contents of the PATHCOM file generated for the sample "employee-prog" applicaton.

```
SET PATHMON BACKUPCPU 1
SET PATHWAY MAXTCPS 10
SET PATHWAY MAXTERMS 10
SET PATHWAY MAXPROGRAMS 10
SET PATHWAY MAXSERVERCLASSES 10
SET PATHWAY MAXSERVERPROCESSES 10
SET PATHWAY MAXSTARTUPS 10
SET PATHWAY MAXPATHCOMS 40
SET PATHWAY MAXASSIGNS 32
SET PATHWAY MAXPARAMS 32
START PATHWAY COLD!

SET TCP PROGRAM $SYSTEM.SYSTEM.PATHTCP2
SET TCP CPUS 0:1
SET TCP MAXTERMS 5
SET TCP MAXSERVERCLASSES 001
SET TCP MAXSERVERPROCESSES 001
SET TCP MAXTERMDATA 06464
SET TCP MAXREPLY    02000
SET TCP NONSTOP 0
SET TCP TCLPROG  $MYVOL.MYSUB.POBJ
ADD TCP ENABLE-TCP

SET PROGRAM TCP ENABLE-TCP
SET PROGRAM TYPE T16-6520 INITIAL EMPLOYEE-PROG
SET PROGRAM TMF OFF
ADD PROGRAM EMPLOYEE-PROG

RESET SERVER ASSIGN, PARAM

SET SERVER PROGRAM $SYSTEM.SYSTEM.ENABLEGS
SET SERVER CPUS 0:1
SET SERVER NUMSTATIC 1
SET SERVER (ASSIGN EMPLOYEE,EMPFILE)
SET SERVER TMF OFF
ADD SERVER ENABLE-SERVER
```

Figure 4-3. PATHCOM Command File for the Sample Employee-Prog Application

You can use the generate PATHCOM command file to establish a PATHWAY system by entering a series of commands that result in:

- The creation of a PATHWAY Monitor (PATHMON) process--PATHMON is the controlling process in a PATHWAY system
- The creation of a log file to which PATHMON can report errors and changes in status
- The assignment of the PATHCTL file--PATHCTL is a disc file where PATHMON maintains status information and the application configuration
- The creation of a PATHCOM process--PATHCOM is the command interface to the PATHMON process

You can place these commands that define, execute, and terminate the application in an edit-type file. Figure 4-4 describes the contents of such a file. Because you can execute the commands in this edit-type file by using the OBEY command, this type of file is called an "obey" file.

```
1.  Purge the current PATHCOM log file and the current
    PATHCTL control file.

       PURGE <log-file>, <control-file>

    For example:

       PURGE log1, enabctl

2.  Create a new PATHCOM log file.

       CREATE <log-file>

    For example:

       CREATE log1

3.  Assign the PATHCTL file to the control-file.

       ASSIGN PATHCTL, <control-file>

    For example:

       ASSIGN PATHCTL, enabctl

4.  Create a PATHMON process.

       PATHMON/NAME <pathmon-name>, NOWAIT,
         CPU <cpu-number>, OUT <log-file>/

    For example:

       PATHMON/NAME $one, NOWAIT, CPU 0, OUT log1/

    Note that <pathmon-names> must not exceed six
    characters (five characters if they are to be used
    across the network), must begin with a dollar sign ($),
    and must be unique within the system (or systems) upon
    which it executes.
```

Figure 4-4. Obey File Commands That Establish a PATHWAY System and Execute an Application (Continued next page)

```
-----------------------------------------------------------------
|                                                               |
|  5.  Cold start PATHWAY using information in the named        |
|      PATHCOM command file.                                    |
|                                                               |
|         PATHCOM/IN <pathcom-file-name>/<pathmon-name>         |
|                                                               |
|      For example:                                             |
|                                                               |
|         PATHCOM/IN singlpth/$one                              |
|                                                               |
|  6.  Run the application.                                     |
|                                                               |
|         PATHCOM <pathmon-name>; RUN <program-name>            |
|                                                               |
|      For example:                                             |
|                                                               |
|         PATHCOM $one;RUN employee-prog                        |
|                                                               |
|  7.  Stop the PATHMON process when all users of the           |
|      application exit from the application.                   |
|                                                               |
|         PATHCOM <pathmon-name>; SHUTDOWN, WAIT                |
|                                                               |
|      For example:                                             |
|                                                               |
|         PATHCOM $one;SHUTDOWN, WAIT                           |
|                                                               |
-----------------------------------------------------------------
```

Figure 4-4. Obey File Commands That Establish a PATHWAY System and Execute an Application (Continued)

To execute the obey file, enter the OBEY command in reponse to the command interpreter prompt as follows:

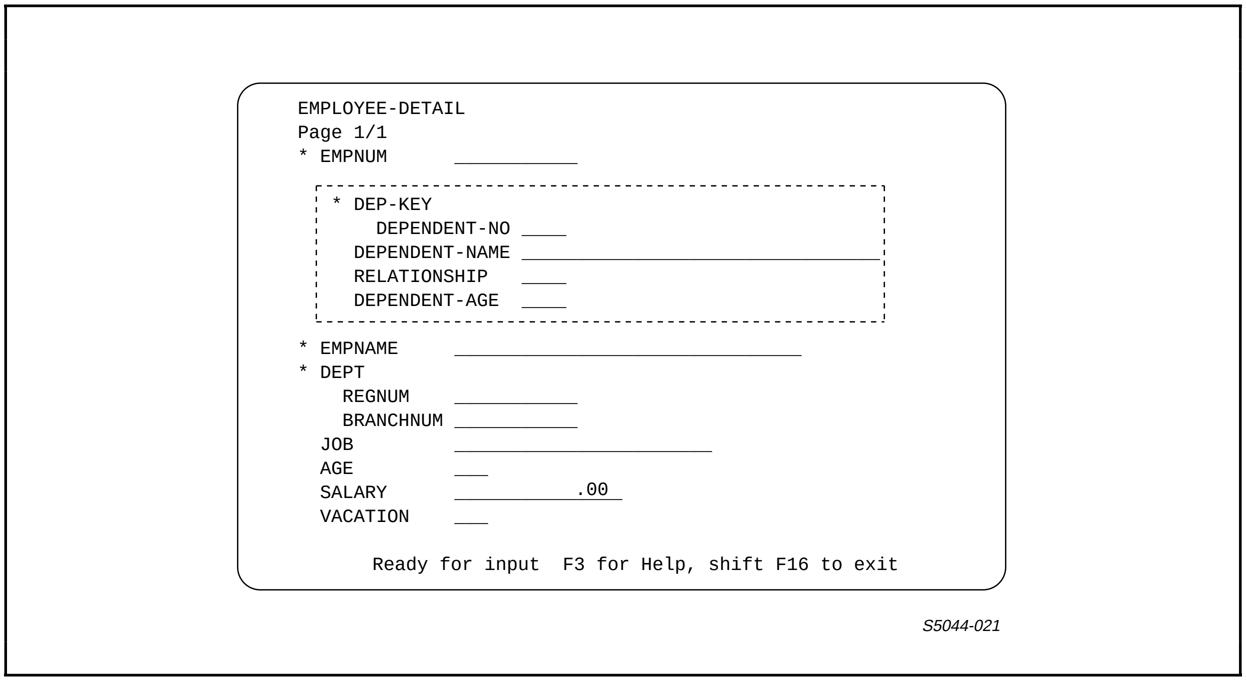
```
   OBEY enabex1
```

where enabex1 is the name of an edit-type file containing the commands described in Figure 4-4.

SECTION 5

CREATING AND RUNNING A MULTIFILE APPLICATION

You can use ENABLE to generate a multifile application through which two or more data base files can be maintained. When you execute the application, a screen similar to the one shown in Figure 5-1 appears.



```
EMPLOYEE-DETAIL
Page 1/1
* EMPNUM      ___________

  * DEP-KEY
      DEPENDENT-NO ____
    DEPENDENT-NAME ________________________________
    RELATIONSHIP   ____
    DEPENDENT-AGE  ____

* EMPNAME     _______________________________
* DEPT
    REGNUM    ___________
    BRANCHNUM ___________
  JOB         _______________________
  AGE         ___
  SALARY      ___________.00_
  VACATION    ___

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

Figure 5-1. Sample Screen Displayed by Employee-Detail Application

The application that displays this screen could be used to maintain information that pertains to employees and their dependents. Figure 5-2 shows the series of commands used to generate and execute this application. You can generate the same application by entering the commands shown in this figure. (Change the file names that appear in lowercase letters, if necessary, to avoid conflicts with existing file names.) These commands accomplish the following tasks:

1. Create a data dictionary

2. Create two data base files

3. Generate the application

4. Establish a PATHWAY system and execute the application

STEP 1: CREATE A DATA DICTIONARY

```
:EDIT ddlsrc2
  *ADD
     1  RECORD EMPLOYEE.
     2  FILE IS employee KEY-SEQUENCED.
     3  02 EMPNUM       PIC 9(4)  HEADING "Employee Number".
     4  02 EMPNAME      PIC X(18) HEADING "Employee Name".
     5  02 DEPT                   HEADING "Department".
     6    04 REGNUM     PIC 9(2)  HEADING "Region Number".
     7    04 BRANCHNUM  PIC 9(2)  HEADING "Branch Number".
     8  02 JOB          PIC X(12) HEADING "Job Title".
     9  02 AGE          PIC 9(2)  HEADING "Age".
    10  02 SALARY       PIC 9(4)V99 HEADING "Salary".
    11  02 VACATION     PIC S99   HEADING "Vacation".
    12  KEY 0 IS EMPNUM.
    13  KEY "en" IS EMPNAME.
    14  KEY "dp" IS DEPT.
    15  END
    16
    17  RECORD DEPENDENTS.
    18  FILE IS depend KEY-SEQUENCED.
    19  02 DEP-KEY.
    20    04 EMP-NO       PIC 9(4)
    21                    HEADING "Employee Number".
    22    04 DEPENDENT-NO PIC 9(2)
    23                    HEADING "Dependent ID".
    24  02 DEPENDENT-NAME PIC X(18)
    25                    HEADING "Dependent Name".
    26  02 RELATIONSHIP   PIC A.
    27  02 DEPENDENT-AGE  PIC 9(2).
    28  KEY 0 IS DEP-KEY.
    29  KEY "dn" IS DEPENDENT-NAME.
    30  END
  *EXIT

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

STEP 2: CREATE THE DATA BASE FILES

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

Figure 5-2. Commands That Generate and Execute the Employee-Detail Application (Continued Next Page)

```
STEP 3:  GENERATE THE APPLICATION

:EDIT enabsrc1
  *ADD
     1  SET BOX RECORD employee
     2  ADD BOX employee
     3  SET BOX RECORD dependents
     4  ADD BOX dependents
     5  SET APPL TREE (01 employee
     6                   02 dependents
     7                   LINK empnum TO OPTIONAL emp-no)
     8  SET APPL PATHCOMFILE multipth
     9  ADD APPL employee-detail
    10  GENERATE *
    11  //
  *EXIT

:ENABLE/IN enabsrc1/


STEP 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 EMPLOYEE-DETAIL
    7  PATHCOM $mult;SHUTDOWN,WAIT
    8  //
  *EXIT

:OBEY enabex2
```

Figure 5-2. Commands that Generate and Execute the
Employee-Detail Application (Continued)

SPECIAL CONSIDERATIONS FOR MULTIFILE APPLICATIONS

Before you try to generate a multifile application, consider the answers to the following questions:

1. Do the files contain matching fields?

2. Do the files contain related information?

Do Matching Fields Exist?

ENABLE can generate an application that uses several data base files only if the record descriptions of the files contain matching fields. Fields can match only when they have compatible data types. Consider, for example, the record descriptions in Figure 5-3 of the two data base files used by the sample "employee-detail" application.

```
RECORD employee.
FILE IS employee  KEY-SEQUENCED.
02 empnum         PIC 9(4)    HEADING "Employee Number"
02 empname        PIC X(18)   HEADING "Employee Name".
02 dept                       HEADING "Department".
  04 regnum       PIC 9(2)    HEADING "Region Number".
  04 branchnum    PIC 9(2)    HEADING "Branch Number".
02 job            PIC X(12)   HEADING "Job Title".
02 age            PIC 9(2)    HEADING "Age".
02 salary         PIC 9(4)V99 HEADING "Salary".
02 vacation       PIC S99     HEADING "Vacation".
KEY 0 IS empnum.
KEY "en" IS empname.
KEY "dp" IS dept.
END

RECORD dependents.
FILE IS depend    KEY-SEQUENCED.
02 dep-key.
  04 emp-no       PIC 9(4)    HEADING "Employee Number"
  04 dependent-no PIC 9(2)    HEADING "Dependent ID".
02 dependent-name PIC X(18)   HEADING "Dependent Name".
02 relationship   PIC A.
02 dependent-age  PIC 9(2).
KEY 0 IS dep-key.
KEY "dn" IS dependent-name.
END
```

S5044-022

Figure 5-3. Sample Record Descriptions

Notice that the "empnum" field of the "employee" record description matches the "emp-no" field of the "dependents" record description. In this case both fields have the same data type: PIC 9 or numeric. If you are not sure that two fields have compatible data types, refer to the ENABLE Reference Manual for more information.

When fields from two files match, you can use them as join fields to establish a logical connection between the two files. When you generate a multifile application, each file must have at least one join field that can be used to link it to another file. If an application accesses more than two data base files, some files may have more than one join field forming links to other files.

ENABLE has one other requirement for multifile applications: when you link two files, the join field of one of the files must be a key field. In Figure 5-3, both of the prospective join fields ("empnum" and "emp-no") are key fields. "Empnum" is a primary key field and "emp-no" is the leading portion of a group field ("dep-key") that is also a primary key field.

If your files do not have appropriate join fields, you cannot use ENABLE to generate an application to maintain them.

Do the Files Contain Related Information?

When you generate a multifile application, the files being accessed should contain related information.

An ENABLE application uses the join-field value of one file to locate or insert a record in another file. If the join fields do not contain related information, the application might not perform in the manner you expect, even though the values in the join fields are compatible.

Suppose, for example, that you try to generate an application to access a file that contains employee information and a file that contains parts information. Suppose further that the join fields of these files are: "enpnum" (containing a four-digit employee number) and "partnum" (containing a four-digit part number). Although ENABLE could generate this application, you could not use it in any meaningful manner because no logical relationship exists between employee numbers and part numbers.

When your data base files do not contain related information, consider using several single-file applications to access the files.

In the sample "employee-detail" application for files that contain related information, the "employee" file contains general information about each employee; the "depend" file contains information about each employee's dependents. The join fields ("empnum" and "emp-no") each represent an employee identification number. Since these fields represent related data, you can use the "employee-detail" application to record, maintain, or display information about a particular employee.

If you have decided that your data base files contain related data and have appropriate join fields, you are ready to generate and execute your own multifile application. To do this:

1. Use the ENABLE commands to generate the application.

2. Use the PATHCOM command file produced by ENABLE to establish a PATHWAY system. You can then execute the application.

GENERATING A MULTIFILE APPLICATION

To generate a multifile application, perform the following tasks:

1. Describe and add a box for each data base file to be used by the application.

2. Associate the boxes with the application and define the logical connections that exist between the boxes.

3. Identify the name of the file to which the PATHCOM commands are to be written.

4. Name and add the application.

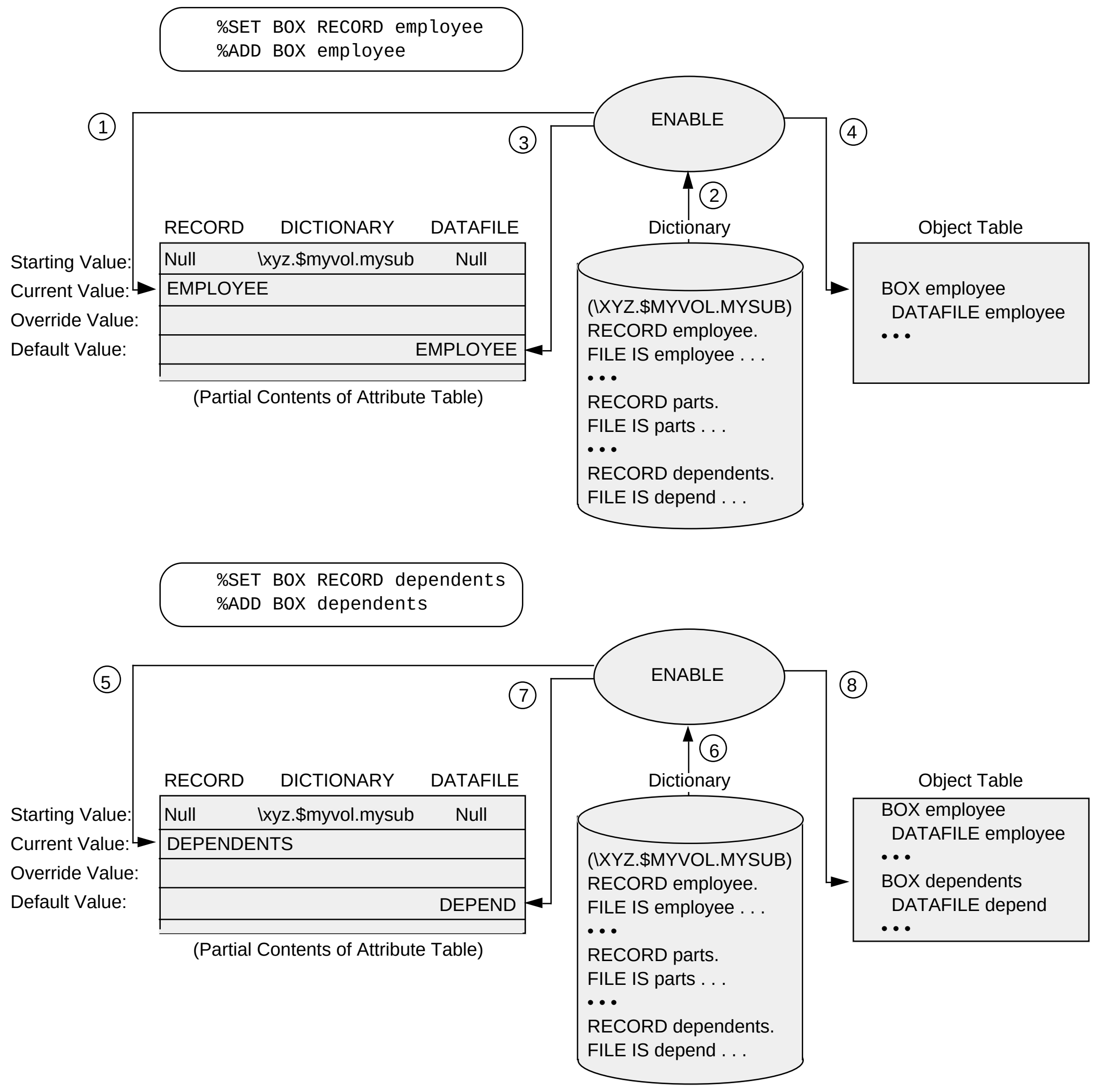
5. Generate the application.

To perform these tasks, use the ENABLE commands. Since some commands are long, you may want to enter them in an edit-type file (called a "command file"). If you enter the commands in a command file, the last task to perform is:

6. Submit the command file to ENABLE.

Describing and Adding Boxes

To describe and add a box for every data base file that the application is to access, proceed much as you did to add a box for a single-file application. Describe a box by supplying a value for the RECORD attribute and add it using the ADD command, as illustrated in Figure 5-4.



Legend

1. ENABLE enters employee as the current value of the RECORD attribute in the attribute table.
2. ENABLE opens the dictionary identified by the DICTIONARY attribute and gets the employee record description.
3. ENABLE uses the file (employee) identified in the record description as the default value of the DATAFILE attribute.
4. ENABLE adds the employee box and its description to the object table.
5. ENABLE changes the current value of the RECORD attribute to dependents.
6. ENABLE opens the dictionary identified by the DICTIONARY attribute and gets the dependents record description.
7. ENABLE uses the file (depend) identified in the dependents record description as the default value of the DATAFILE attribute.
8. ENABLE adds the dependents box and its description to the object table.

S5044-023

Figure 5-4. Describing and Adding Two Boxes

Although you can supply values for other box attributes when you describe and add a box, this discussion assumes that the default values suit your needs. If you use the SET command to describe a box, you might want to use the RESET BOX command (to reset the attributes to their starting values) before you describe and add the next box. After you describe and add the necessary boxes, you are ready to associate the boxes with an application.

Associating and Linking Boxes

When you generate a multifile application, you must identify:

- The boxes to be used by the application
- The logical connections or links that exist between the boxes
- The order in which the boxes are to be connected for the application

To perform this task, supply a value for the TREE application attribute. When you supply a value for this attribute, you build a logical structure called a tree structure that ENABLE uses to organize the application.

What Is a Tree Structure?

A tree is a hierarchical structure similar to a family tree. Within the tree structure, each box has a specific position, much as you have a specific position within your own family. The position of a box within a tree structure is called its level. A box at one level of the tree structure is connected to a box (or boxes) at other levels of the tree structure by links. Figure 5-5 illustrates a sample two-level tree structure and shows the value of the TREE attribute that builds this structure.

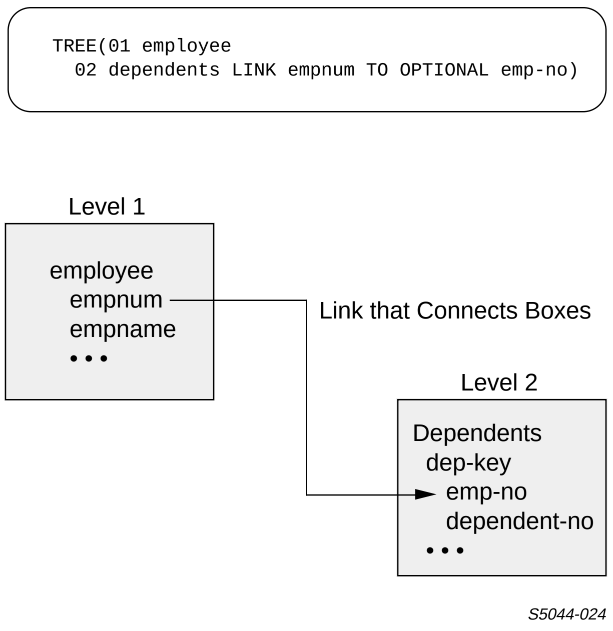


Figure 5-5. Sample Tree Structure

What Is a Link?

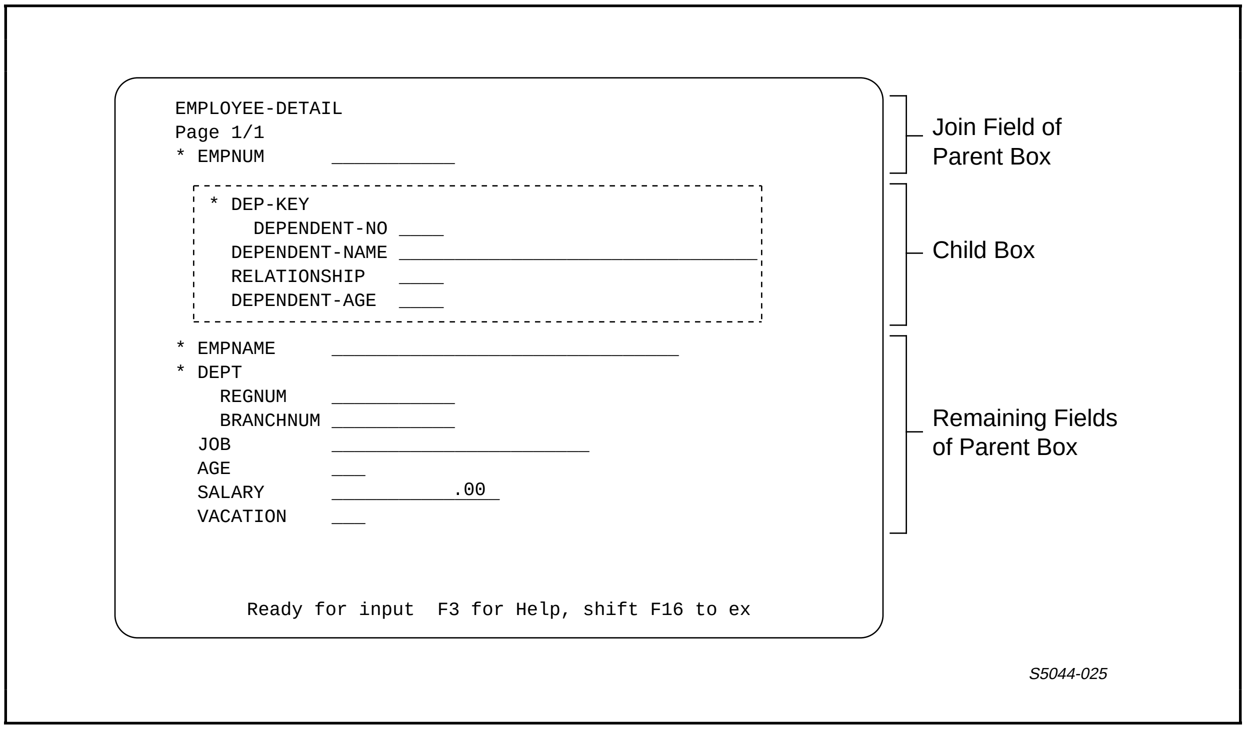
A link is the portion of the tree structure that establishes a logical connection between two boxes. A tree structure might have many links that connect a box to several other boxes; however, only two boxes are connected by any single link.

ENABLE provides a special kind of link called a LINK OPTIONAL. When you establish a LINK OPTIONAL between two boxes, you define the order in which the application can read or insert records for the boxes. This order establishes a dependency between the records associated with both. With a LINK OPTIONAL, the application must use the join-field value of a record from one box to read or insert a record for the other box. Access to a record in the second box depends upon the presence of a record that has a matching join-field value in the first.

Because this dependency is similar to the relationship 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. You can read or insert a record for a child box only if you have already read or inserted a record for the parent box.

The screen displayed by an application reflects the parent-child relationship you define. An application displays a child box after the join field of the parent box. If any other fields in the parent box follow the join field, the application displays these fields below the child box. An application does not display the join field of a child box.

Figure 5-6 shows the screen displayed by the sample "employee-detail" application. For this application, the employee box is the parent box and the "dependents" box is the child box. Notice that the join field ("empnum") of the "employee" box appears before the "dependents" box and that the join field of the dependents box ("emp-no") does not appear on the screen. An ENABLE application does not display the join field of the child box because the data in this field is identical to the data displayed or inserted in the join field of the parent box.



```
EMPLOYEE-DETAIL
Page 1/1
* EMPNUM          ___________

   * DEP-KEY
       DEPENDENT-NO ____
     DEPENDENT-NAME ________________________________
     RELATIONSHIP   ____
     DEPENDENT-AGE  ____

* EMPNAME         _______________________________
* DEPT
    REGNUM        ___________
    BRANCHNUM     ___________
  JOB             _______________________
  AGE             ___
  SALARY          ___________.00
  VACATION        ___

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

Figure 5-6. Parent and Child Boxes on the Screen

When you establish a link between two boxes, you must decide which box is to be the parent box and which box is to be the child.

Choosing a Parent and Child Box

To decide which box is to be a parent box and which box is to be a child, consider the purpose for which the application is being generated. ENABLE imposes two requirements regarding this choice:

1. The join field of a child box must be a primary key field, an alternate key field, a courtesy key field, or the leading portion of a primary or alternate key field.

2. The size of the join field for the parent box must be equal to or less than the size of the join field of the child box.

Refer to the ENABLE Reference Manual for more information about these requirements.

After you determine the links that can exist between the boxes used by your application, you must identify the level at which each box resides in the tree structure.

The Levels of the Tree Structure

A simple tree structure, such as the one you build for a single-file application, has one level and a single box that resides at that level. A more complex structure, such as one that you build for an application that accesses several files, can have many levels, with several boxes residing at each level except for the first. At this, lowest, level there can be only one box. ENABLE imposes no restrictions on the number of boxes that reside at other levels of the tree structure and has a maximum of 50 levels per application.

Before you identify the level at which each box resides, you should understand how they affect the generated application. When you identify the levels of your tree structure:

1. Only one box can reside at the first level. A box that resides at this level must have one or more join fields that you can use to link it to a child box (or boxes) at the second level of the tree.

2. For a box at any level except the first, a join field must exist that links that box to a parent box at the next lower level of the tree structure.

3. If you link two or more boxes to the same parent box, these boxes must reside at the same level of the tree structure.

The level at which a box resides in the tree structure affects the appearance of the screen displayed by the application as follows:

- A field from the box at the first level of the tree must appear as the first field on the screen.

- For each subsequent level, a field from a box at that level must appear before any fields from boxes at higher levels.

Before you determine the levels at which your boxes are to reside, consider the purpose for which you are generating the application, the possible links that you can establish between boxes, and the logical relationship between these boxes.

If your application uses only two boxes, you can identify either box as residing at the first level of the tree structure.

If your application uses three or more boxes, the process of determining appropriate levels becomes more complicated. Suppose, for example, that you want to generate an application that uses three boxes: "employ-box," "depends-box," and "insure-box." Assume that these boxes represent the following information:

- "Employ-box" represents general information about employees.

- "Depends-box" represents information about the dependents of these employees.

- "Insure-box" represents information about each employee's insurance coverage. This information also indicates whether dependents are covered by the employee's insurance.

Now suppose that each of these boxes has a field that represents an employee identification number. Depending on the purpose for which you are generating this application, you could build a variety of tree structures with a variety of links. Figure 5-7 illustrates the various tree structures which you could build for these boxes.

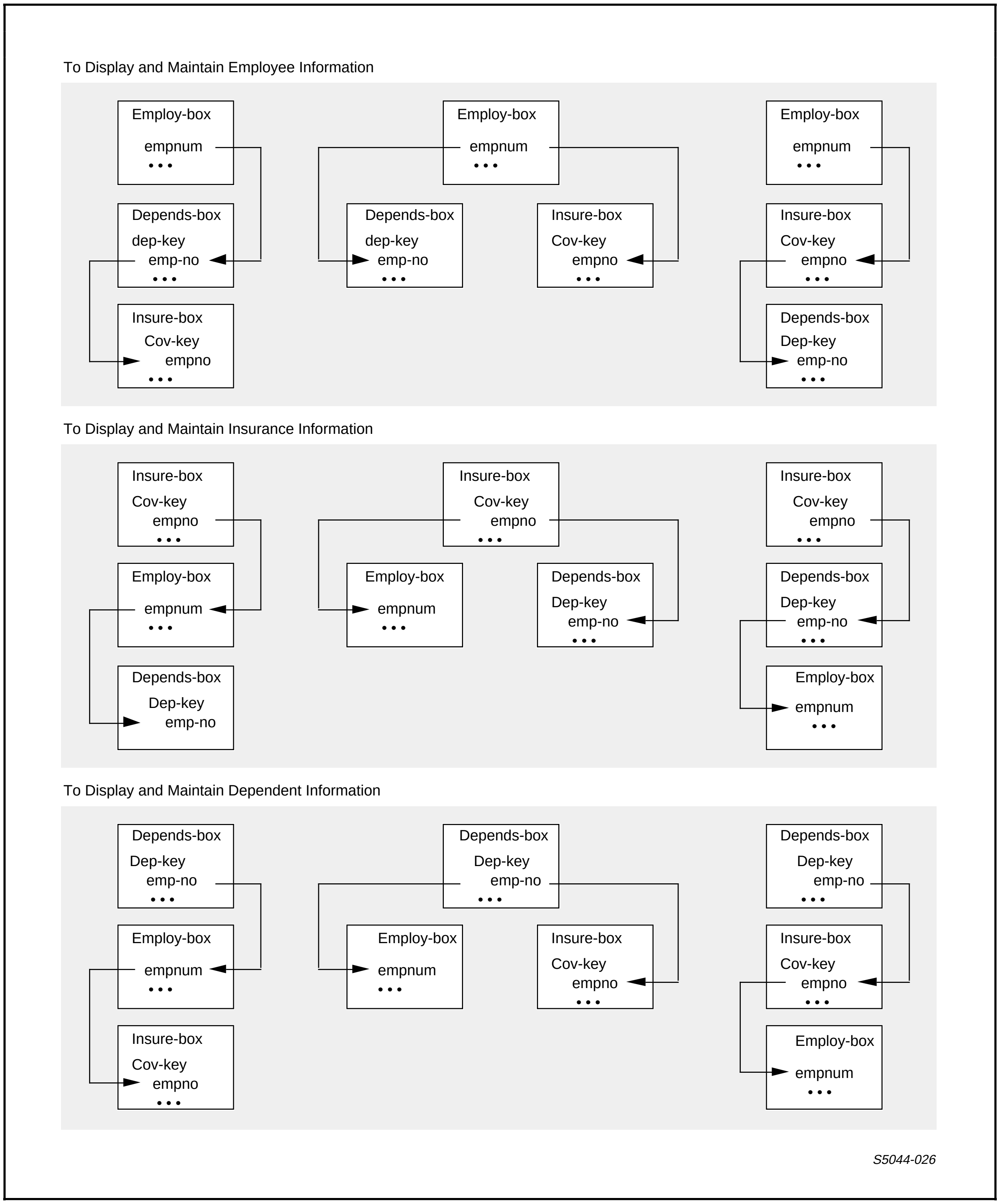


Figure 5-7. Box Levels When Multiple Links Are Possible

Sometimes the links that you can establish will determine the level at which a box must reside. Suppose that your application uses a box named "dept-box" as well as two of the boxes ("employ-box" and "depends-box") described earlier. Suppose further that you can link "dept-box" to "employ-box" but you cannot link "dept-box" to "depends-box." Depending on the purpose of the application, you could identify the levels shown in Figure 5-8.

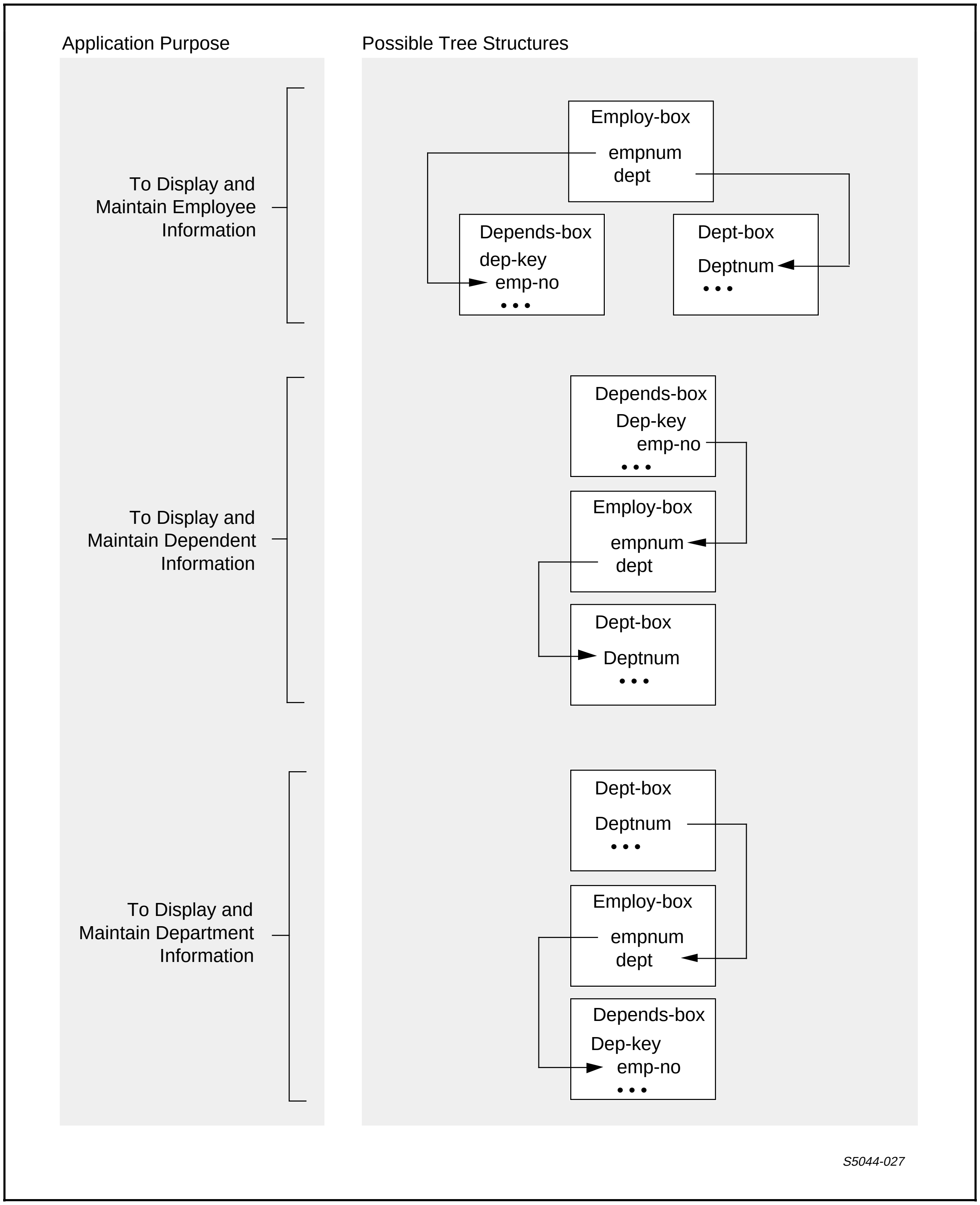


Figure 5-8. Box Levels When Links Are Limited

After you identify the level at which a box is to reside, you are ready to supply a value for the TREE attribute.

Supplying a Value for the TREE Attribute

Although you can supply a value for the TREE attribute in several ways, the examples in this discussion use the SET APPL command to do so.

To supply values for the TREE attribute, you must:

- Associate a box with a level number
- Provide a LINK OPTIONAL clause for each box except the box at the first level of the tree structure

Associating a Box With a Level Number. To define the level at which a box resides in the tree, you associate the box name with a level number (similar to a DDL level number). Valid values for level numbers range from 1 to 50.

To define a box as being at the first level of the tree, specify a level number that is lower in value than any other level number in the tree. For example, consider the following box names and associated level numbers:

```
06 employ-box 08 depends-box 08 coverage-box 10 location-box
```

The level numbers associated with these box names indicate that:

- "Employ-box" is at the first level of the tree structure
- "Depends-box" and "coverage-box" are at the second level of the tree structure
- "Location-box" is at the third level of the tree structure

Level numbers do not have to correspond to the precise level at which a box resides, nor do they have to be sequential. These rules govern level numbers for the TREE attribute:

1. The numerically lowest-level number identifies the box at the first level of the tree; only one box can have this level number.

2. The level number of a parent box must be lower than the level number of its child box (or boxes).

3. The level numbers for all child boxes associated with the same parent box must be the same.

4. Valid level numbers range between 1 and 50.

Using the LINK OPTIONAL Parameter. When you supply a value for the TREE attribute, you must use the LINK OPTIONAL parameter with every box except the box at the first level of the tree. When you use this parameter, you identify the link that connects a child box to a parent box.

The LINK OPTIONAL parameter has two forms. You use the first form if the join field of the parent box and the join field of the child box have different names. You use the second form if the join fields of both boxes have the same name.

If you use the first form of the LINK OPTIONAL parameter, you enter the join-field name of the parent box before the join-field name of the child box, for example:

```
   SET APPL TREE (01 employee
                  02 dependents LINK empnum
                  TO OPTIONAL emp-no)
```

In this case:

- "employee" is the name of the parent box.

- "dependents" is the name of the child box.

- "empnum" is the join field of the parent record.

- "emp-no" is the join field of the child record.

If you use the second form of the LINK OPTIONAL parameter, you enter the name of the parent box, the name of the child box, and the common name of the join fields from both boxes, for example:

```
   SET APPL TREE (01 region
                     02 branch
                     LINK region TO OPTIONAL branch
                     VIA regnum)
```

In this case:

- "region" is the parent box.
- "branch" is the child box.
- "regnum" is the join field in both parent and child records.

Effect of the Tree Structure on a Generated Application

When you build a tree structure for your application, that structure affects the following aspects of a generated application:

- The screen displayed by the application, as follows:

  --A field from the box at the first level always appears as the field on the screen.

  --The join field of a parent box always appears on the screen immediately before any child box (or boxes).

  --A join field of a child box does not appear on the terminal screen.

- The records that can be read and inserted for each box: The organization of the tree structure determines the access path that the generated application follows to read and insert records.

The following paragraphs describe two sample tree structures and discuss how these tree structures affect the preceding aspects of their respective applications.

<u>Sample Two-Level Tree Structure</u>. Figure 5-9 shows an example of a two-level tree structure. This figure also shows the SET TREE command that builds this tree structure.

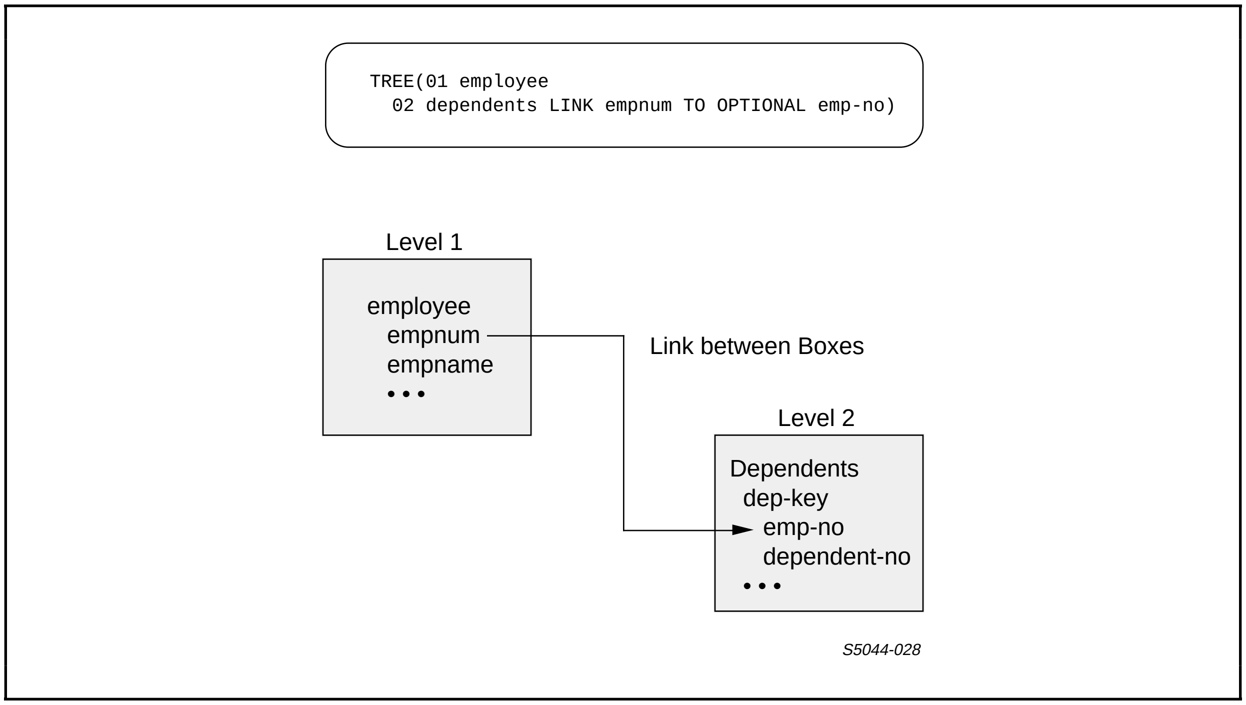
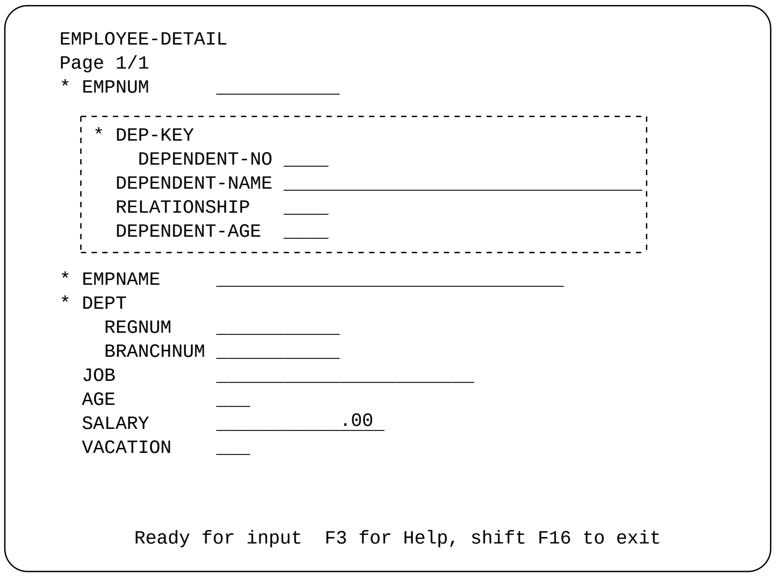

```
TREE(01 employee
  02 dependents LINK empnum TO OPTIONAL emp-no)
```

Figure 5-9. Sample Two-Level Tree Structure

Figure 5-10 shows the screen displayed by an application generated with this tree structure. Note that a field from the "employee" box (the box at the first level of the tree structure) appears first on the screen. The "dependents" box directly follows the join field ("empnum") of the "employee" box. The join field ("emp-no") of the "dependents" box does not appear on the screen.



```
EMPLOYEE-DETAIL
Page 1/1
* EMPNUM        ___________

  * DEP-KEY
      DEPENDENT-NO ____
    DEPENDENT-NAME ________________________________
    RELATIONSHIP   ____
    DEPENDENT-AGE  ____

* EMPNAME       _______________________________
* DEPT
    REGNUM      ___________
    BRANCHNUM   ___________
  JOB           _______________________
  AGE           ___
  SALARY        ___________.00_
  VACATION      ___

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

S5044-029

Figure 5-10. Sample Screen With Two Boxes

The tree structure defined for an application also determines the the records that the application can retrieve or insert for each box. If the application can retrieve or insert a record for a box, that record is said to qualify for the application-access path.

An ENABLE application can read or insert any record for the box at the first level of the tree structure. Therefore, all the records associated with this box qualify for the application-access path.

An ENABLE application can read or insert records for a box at the second level of the tree only if those records match (have the same join-field values as) records read for or inserted in the box at the first level of the tree. Therefore, a record for a box at the second level of the tree qualifies for the application-access path only if that record matches a record for the box at the first level of the tree.

Figure 5-11 illustrates this concept for the "employee" and "dependents" boxes.

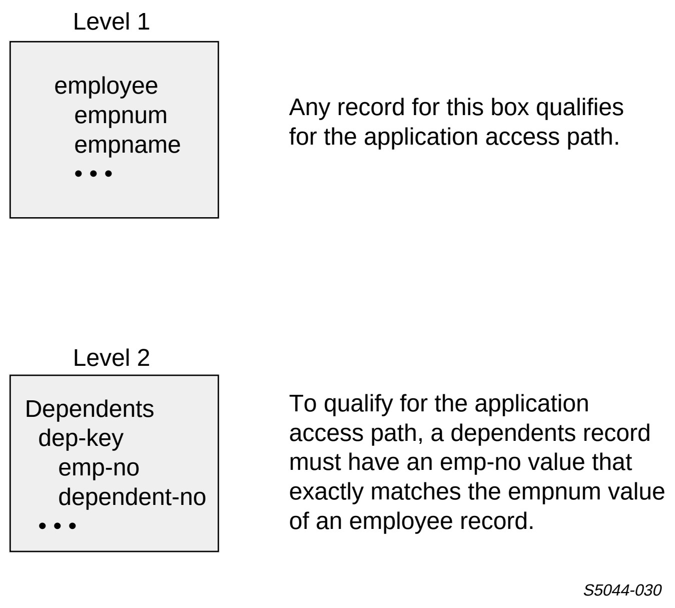
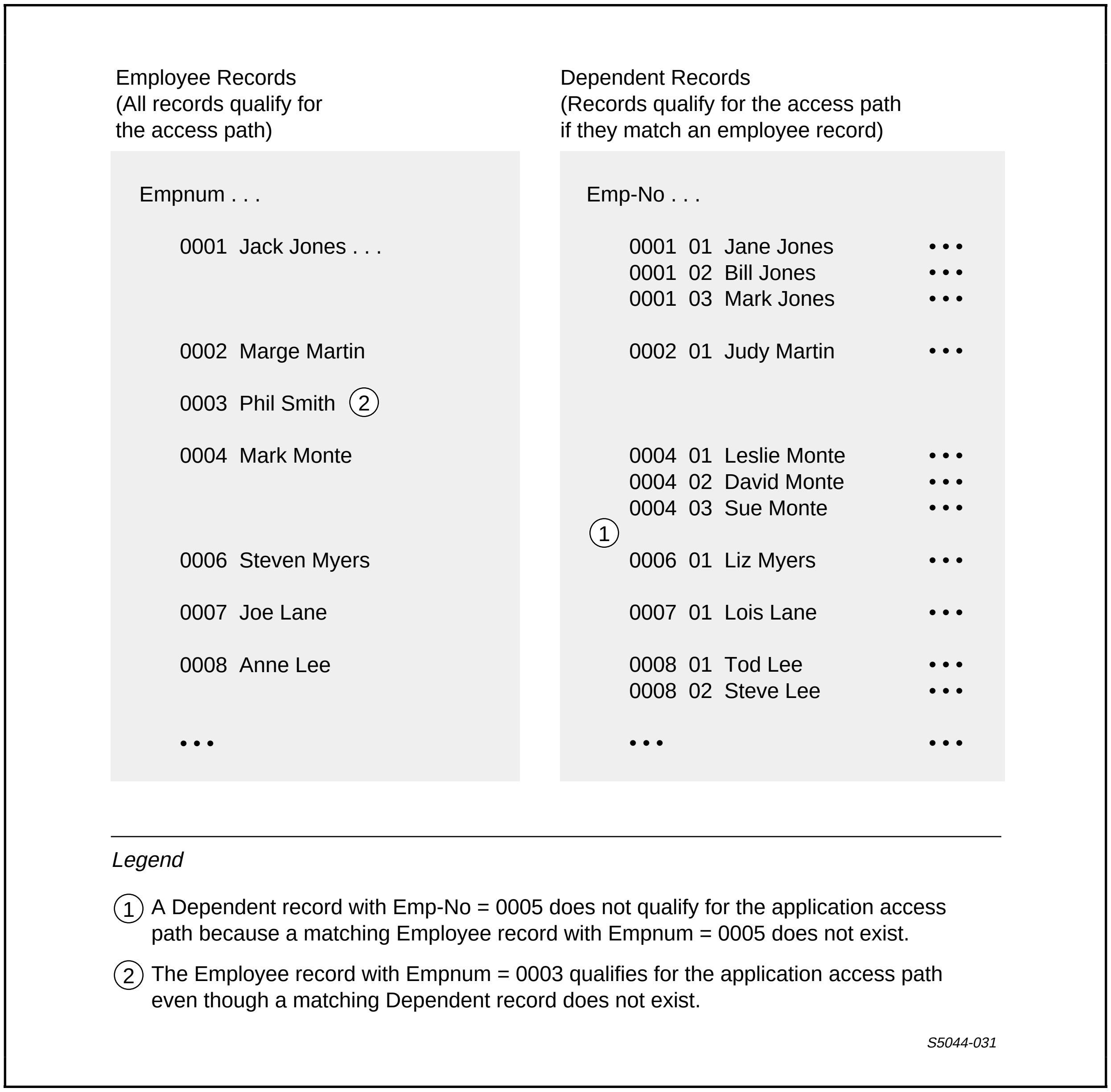

Level 1
employee
empnum
empname
• • •
Any record for this box qualifies for the application access path.
Level 2
Dependents
dep-key
emp-no
dependent-no
• • •
To qualify for the application access path, a dependents record must have an emp-no value that exactly matches the empnum value of an employee record.
S5044-030

Figure 5-11. Application-Access Path for the Employee-Detail Application

Figure 5-12 shows the partial contents of some sample records associated with the "employee" and "dependents" boxes. This figure also identifies the records from both boxes that qualify for the application-access path.



Employee Records
(All records qualify for
the access path)

Dependent Records
(Records qualify for the access path
if they match an employee record)

Empnum . . .	Emp-No . . .	
0001 Jack Jones . . .	0001 01 Jane Jones	• • •
	0001 02 Bill Jones	• • •
	0001 03 Mark Jones	• • •
0002 Marge Martin	0002 01 Judy Martin	• • •
0003 Phil Smith (2)		
0004 Mark Monte	0004 01 Leslie Monte	• • •
	0004 02 David Monte	• • •
	0004 03 Sue Monte	• • •
	(1)	
0006 Steven Myers	0006 01 Liz Myers	• • •
0007 Joe Lane	0007 01 Lois Lane	• • •
0008 Anne Lee	0008 01 Tod Lee	• • •
	0008 02 Steve Lee	• • •
• • •	• • •	• • •

Legend

(1) A Dependent record with Emp-No = 0005 does not qualify for the application access path because a matching Employee record with Empnum = 0005 does not exist.

(2) The Employee record with Empnum = 0003 qualifies for the application access path even though a matching Dependent record does not exist.

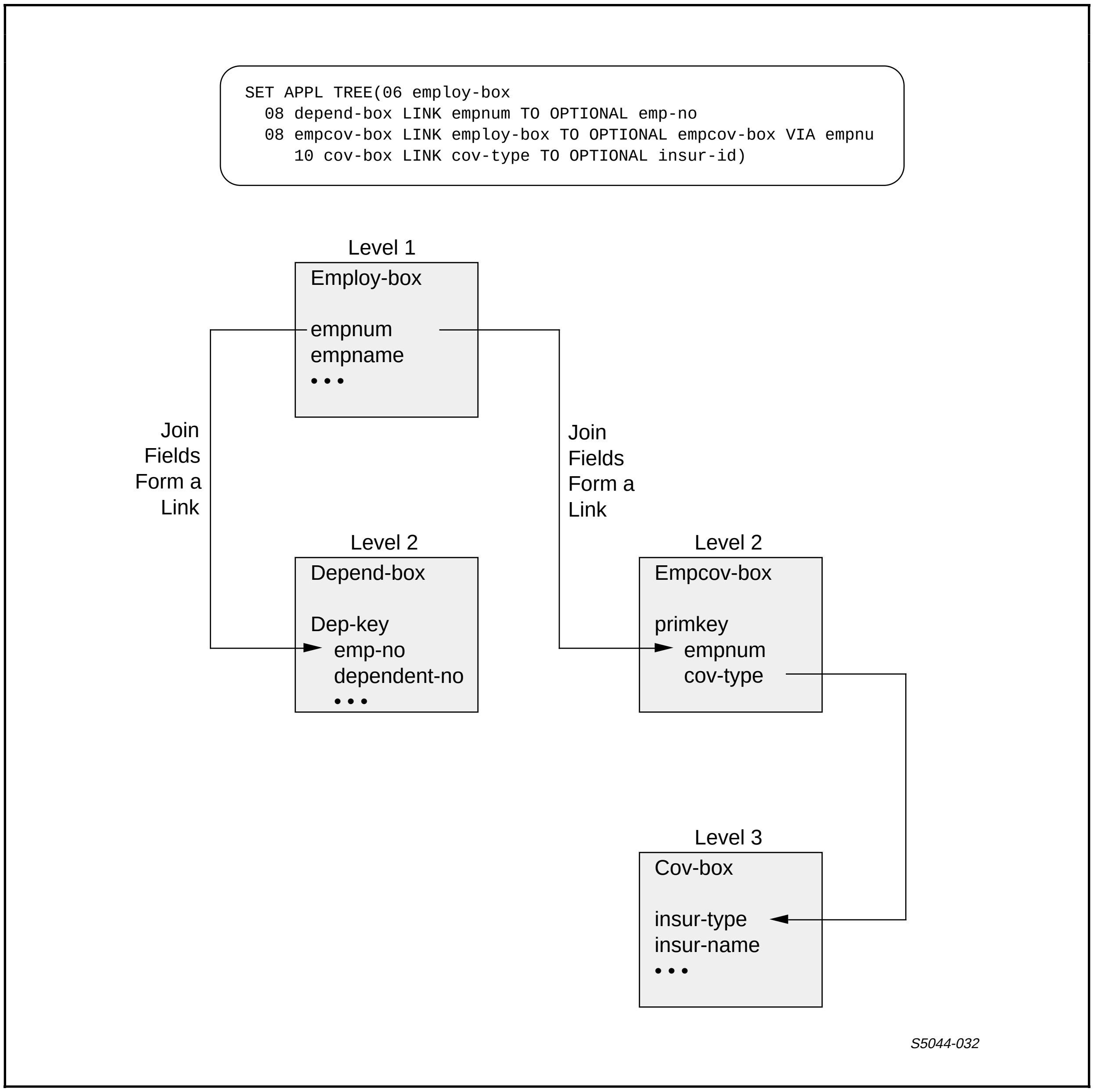
Figure 5-12. Qualified Records for a Tree Structure With Two Boxes

The access path established for an application also affects the way that you can perform delete and update operations on records for these boxes, for example:

- If you update a parent record and change the value of its join field, the child record that previously matched the updated record no longer qualifies for the application-access path.

- If you delete a parent record before you delete a matching child record, you can no longer retrieve the child record. Since the matching parent record no longer exists, the child record does not qualify for the application-access path.

Sample Three-Level Tree Structure. Figure 5-13 shows a three-level tree structure and the SET TREE command that builds this structure.



```
SET APPL TREE(06 employ-box
  08 depend-box LINK empnum TO OPTIONAL emp-no
  08 empcov-box LINK employ-box TO OPTIONAL empcov-box VIA empnu
     10 cov-box LINK cov-type TO OPTIONAL insur-id)
```

Figure 5-13. Sample Three-Level Tree Structure

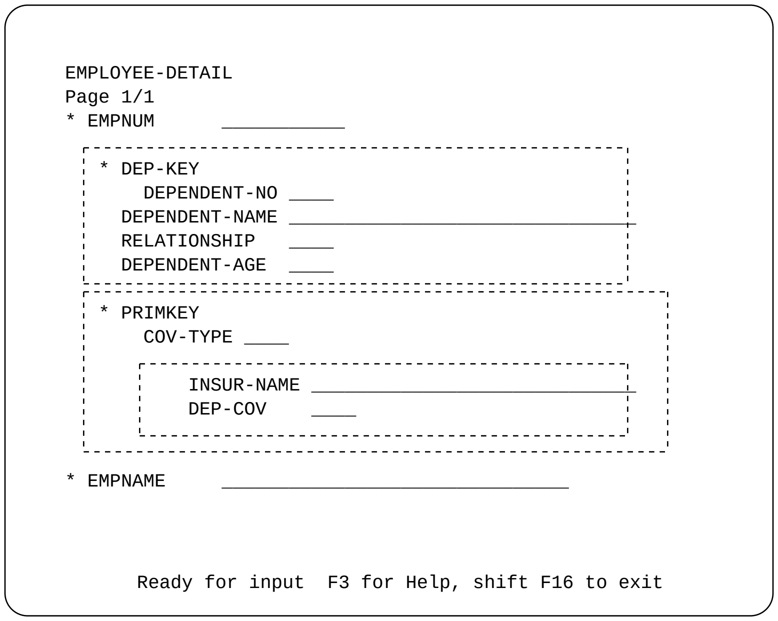
Figure 5-14 shows the screen displayed by an application generated with this tree structure. Note the following:

1. A label ("empnum") from "employ-box" appears first on the screen.

2. "Depend-box" appears directly after "empnum" (the join field of "employ-box"). The join field ("emp-no") of "depend-box" does not appear.

3. The second child box, "empcov-box," follows "depend-box." Note that both "depend-box" and "empcov-box" begin in the same screen column because they are both children of "employ-box." The join field ("empnum") of "empcov-box" does not appear.

4. "Cov-box," the child of "empcov-box," appears to be nested in "empcov-box." The join field ("insur-type") of "cov-box" does not appear.



```
EMPLOYEE-DETAIL
Page 1/1
* EMPNUM      ___________

 * DEP-KEY
     DEPENDENT-NO ____
   DEPENDENT-NAME _______________________________
   RELATIONSHIP   ____
   DEPENDENT-AGE  ____

 * PRIMKEY
     COV-TYPE ____

         INSUR-NAME _____________________________
         DEP-COV    ____

* EMPNAME     _______________________________

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

Figure 5-14. Screen With Four Boxes

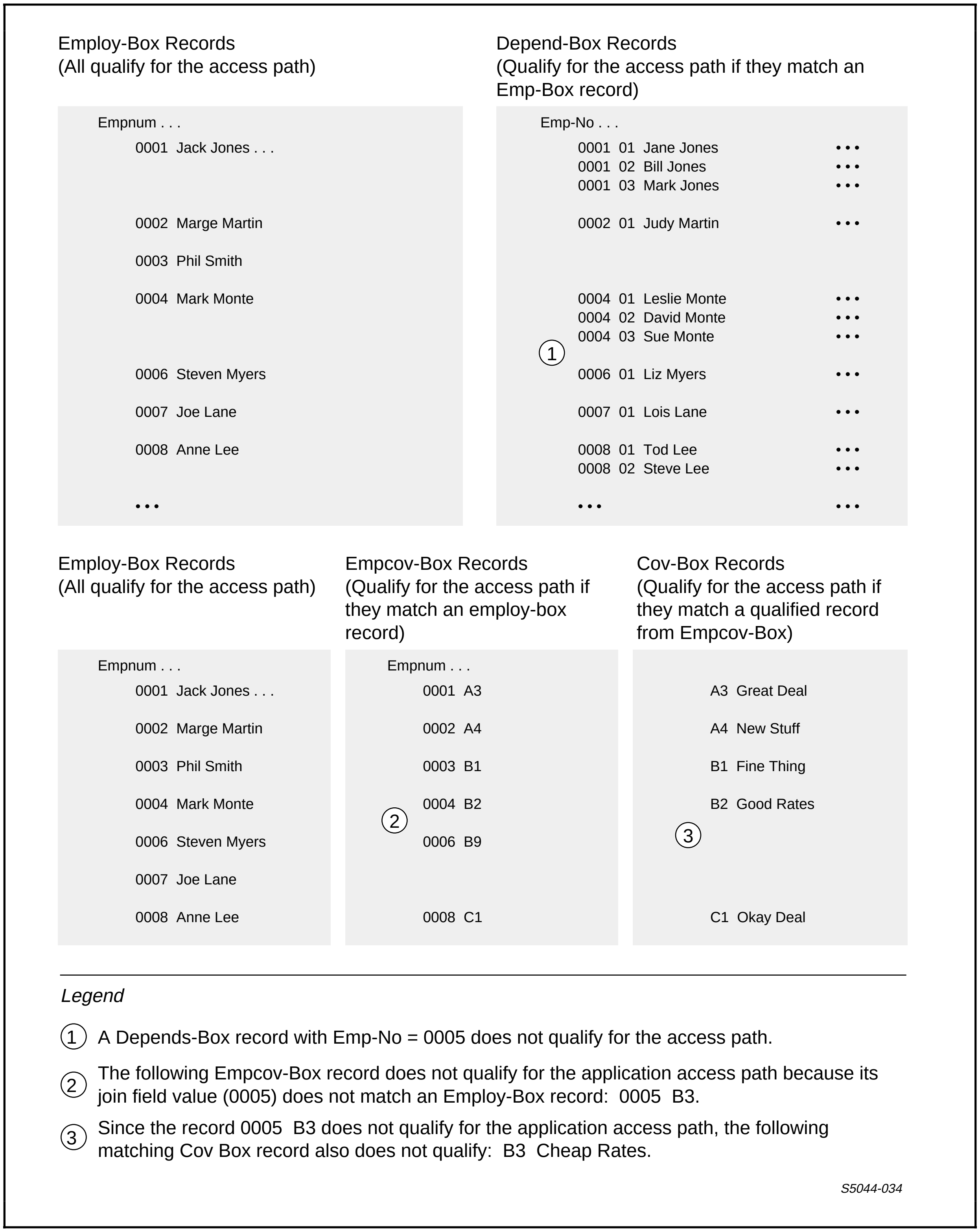
Figure 5-15 shows the partial contents of the files associated with the application that displays the screen shown in Figure 5-14. This figure also shows the records from each file that qualify for the application-access path.

When the tree structure of an application has more than two levels, records qualify for the application-access path as follows:

1. The application can read or insert any record for the box at the first level of the tree. Therefore, all records for this box qualify for the application-access path.

2. The application can read or insert a record for a box at the second level of the tree only if that record matches (has the same join-field value) a record for a box at the first level of the tree. Therefore, to qualify for the application-access path, a record at the second level of the tree must match any record at the first level of the tree.

3. The application can read or insert a record for a box at the third level of the tree only if that record matches (has the same join-field value) a qualified record for a box at the second level of the tree. Therefore, to qualify for the application-access path, a record at the third level of the tree must match a qualified record at the second level of the tree.



Employ-Box Records
(All qualify for the access path)

Empnum . . .
0001 Jack Jones . . .
0002 Marge Martin
0003 Phil Smith
0004 Mark Monte
0006 Steven Myers
0007 Joe Lane
0008 Anne Lee
• • •

Depend-Box Records
(Qualify for the access path if they match an Emp-Box record)

Emp-No . . .	
0001 01 Jane Jones	• • •
0001 02 Bill Jones	• • •
0001 03 Mark Jones	• • •
0002 01 Judy Martin	• • •
0004 01 Leslie Monte	• • •
0004 02 David Monte	• • •
0004 03 Sue Monte	• • •
0006 01 Liz Myers	• • •
0007 01 Lois Lane	• • •
0008 01 Tod Lee	• • •
0008 02 Steve Lee	• • •
• • •	• • •

Employ-Box Records
(All qualify for the access path)

Empcov-Box Records
(Qualify for the access path if they match an employ-box record)

Cov-Box Records
(Qualify for the access path if they match a qualified record from Empcov-Box)

Empnum . . .	Empnum . . .	
0001 Jack Jones . . .	0001 A3	A3 Great Deal
0002 Marge Martin	0002 A4	A4 New Stuff
0003 Phil Smith	0003 B1	B1 Fine Thing
0004 Mark Monte	0004 B2	B2 Good Rates
0006 Steven Myers	0006 B9	
0007 Joe Lane		
0008 Anne Lee	0008 C1	C1 Okay Deal

Legend

1 A Depends-Box record with Emp-No = 0005 does not qualify for the access path.

2 The following Empcov-Box record does not qualify for the application access path because its join field value (0005) does not match an Employ-Box record: 0005 B3.

3 Since the record 0005 B3 does not qualify for the application access path, the following matching Cov Box record also does not qualify: B3 Cheap Rates.

S5044-034

Figure 5-15. Qualified Records for an Application With a Three-Level Tree Structure

Supplying a value for the TREE attribute is the most complex task that you must perform to generate a multifile application. The remaining tasks are simple to perform. These tasks are:

- Identifying the name of the PATHCOM command file
- Naming and adding the application
- Generating the application
- Submitting the command file to ENABLE

Identifying the Name of the PATHCOM Command File

To complete the description of the application, request a PATHCOM file and identify its name. To request a PATHCOM file, supply the file name as the value of the PATHCOMFILE attribute. For example, the following SET APPL command requests that ENABLE write the PATHCOM command file to a file named "multipth":

```
SET APPL PATHCOMFILE multipth
```

Naming and Adding the Application

You can now name and add the application and its description to the object table. To add an application, you use the ADD APPL command. For example, the following ADD APPL command adds the sample "employee-detail" application:

```
ADD APPL employee-detail
```

Generating an Application

To generate an application, you use the GENERATE command. Since you have already added an application to the object table, you can generate your application by using the following form of the GENERATE command:

```
GENERATE APPL *
```

When you use this form of the GENERATE command, ENABLE generates all the applications that exist in the object table.

Submitting the Command File to ENABLE

If you have entered your ENABLE commands on a command file, you complete the tasks involved in generating an application by submitting the command file to ENABLE. To submit the command file, you use the following form of the ENABLE run command:

```
   ENABLE/ IN <command-file> /
```

<command-file> is your edit-type file of ENABLE commands.

You enter this command in response to the command interpreter prompt (:), for example:

```
  :ENABLE/IN enabcmds/
```

ESTABLISHING A PATHWAY SYSTEM AND EXECUTING THE APPLICATION

An ENABLE application executes within a PATHWAY system. If you supply a value for the PATHCOMFILE attribute when you generate an application, ENABLE writes PATHCOM commands to the named EDIT file. You can use these commands to establish a PATHWAY system to execute your application. Figure 5-16 shows the PATHCOM commands written by ENABLE for the sample "employee-detail" application.

```
SET PATHMON BACKUPCPU 1
SET PATHWAY MAXTCPS 10
SET PATHWAY MAXTERMS 10
SET PATHWAY MAXPROGRAMS 10
SET PATHWAY MAXSERVERCLASSES 10
SET PATHWAY MAXSERVERPROCESSES 10
SET PATHWAY MAXSTARTUPS 10
SET PATHWAY MAXPATHCOMS 40
SET PATHWAY MAXASSIGNS 32
SET PATHWAY MAXPARAMS 32
START PATHWAY COLD!

SET TCP PROGRAM $SYSTEM.SYSTEM.PATHTCP2
SET TCP CPUS 0:1
SET TCP MAXTERMS 5
SET TCP MAXSERVERCLASSES 002
SET TCP MAXSERVERPROCESSES 002
SET TCP MAXTERMDATA 07560
SET TCP MAXREPLY    02000
SET TCP NONSTOP 0
SET TCP TCLPROG  $MKT.SAMPLE.POBJ
ADD TCP ENABLE-TCP

SET PROGRAM TCP ENABLE-TCP
SET PROGRAM TYPE T16-6520 INITIAL EMPLOYEE-DETAIL
SET PROGRAM TMF OFF
ADD PROGRAM EMPLOYEE-DETAIL

RESET SERVER ASSIGN, PARAM

SET SERVER PROGRAM $SYSTEM.SYSTEM.ENABLEGS
SET SERVER CPUS 0:1
SET SERVER NUMSTATIC 1
SET SERVER (ASSIGN EMPLOYEE,$MKT.SAMPLE.EMPLOYEE)
SET SERVER (ASSIGN DEPENDENTS,$MKT.SAMPLE.DEPEND)
SET SERVER TMF OFF
ADD SERVER ENABLE-SERVER
```

Figure 5-16. PATHCOM Command File for the Employee-Detail Application

To use the PATHCOM command file to establish a PATHWAY system for a multifile application, enter a series of commands similar to those that you enter to establish a PATHWAY system for a single-file application. These commands result in the following:

- The creation of a PATHWAY Monitor (PATHMON) process--PATHMON is the controlling process in a PATHWAY system

- The creation of a log file to which PATHMON can report errors and changes in status

- The assignment of the PATHCTL file--PATHCTL is a disc file where PATHMON maintains status information and the application configuration

- The creation of a PATHCOM process--PATHCOM is the command interface for the PATHWAY system

You can place these commands in a single obey file for convenient definition, execution, and subsequent termination of the ENABLE application. Figure 5-17 shows an annotated example of an obey file that establishes a PATHWAY system and executes the sample "employee-detail" application.

```
PURGE multilog, multictl                                    (1)
CREATE multilog                                             (2)
ASSIGN PATHCTL, multictl                                    (3)
PATHMON/NAME $mult, NOWAIT, CPU 0, OUT multilog/            (4)
PATHCOM/IN multipth/$mult                                   (5)
PATHCOM $mult; RUN employee-detail                          (6)
PATHCOM $mult; SHUTDOWN, WAIT                               (7)
```

NOTES

(1) Purges the current PATHCOM log file and the current PATHCTL file.

(2) Creates a new PATHCOM log file.

(3) Assigns the PATHCTL file to "multictl."

(4) Creates a PATHMON process. PATHMON names must not exceed six characters (five characters if PATHMON is to be used across the network), must begin with a dollar sign ($), and must be unique within the system upon which PATHWAY executes.

(5) Cold starts PATHWAY using the generated PATHCOM command file.

(6) Runs the application.

(7) Stops the PATHMON process when you exit from the application.

Figure 5-17. Obey File Commands That Establish a PATHWAY System and Execute the Employee-Detail Application

To execute the obey file, use the OBEY command. If the name of the edit-type file that contains these commands is "enabex2," you can execute this file by entering the following in response to the command interpreter prompt:

```
:OBEY enabex2
```

SECTION 6

TAILORING AN APPLICATION

Sections 4 and 5 describe the tasks involved in generating a basic application--an application that displays a standard screen and that can perform delete, insert, read, and update operations on the data base files to which it has access. This section describes optional capabilities with which you can:

- Provide a customized screen for an application. You might, for example, want to provide a customized screen that improves the usability of an application and enhances its appearance.

- Indicate the operations (DELETE, INSERT, READ, or UPDATE) that can be performed by an application. You might want to limit your application to specific operations, for example, to protect your data base files from inadvertent changes.

- Specify a method that the application must use to ensure the integrity of the data base files. You might, for example, need to indicate that your data base files are audited by the Transaction Monitoring Facility (TMF).

PROVIDING A CUSTOMIZED SCREEN

While a standard screen format is satisfactory for many applications, you may want to provide a customized screen in order to enhance the appearance of the screen and improve the usability of the application.

General Guidelines for Customizing a Screen

Before you decide on a format for your customized screen, consider the needs of the people who will use it. When choosing a screen format for an application, do the following wherever possible:

- Provide a descriptive screen title. If you supply a descriptive screen title, the users can be confident that they are using the appropriate application.
- Provide your own labels for screen fields. You can supply labels that are more informative than the field names used for a standard format screen.
- Display a key field as the first field on the screen. Most users expect to be able to enter a value in the first screen field, press an appropriate read function key, and retrieve a record. If the first screen field is not a key field, the user must tab to a key field before retrieving a record.
- Provide user information within a box whenever necessary. If appropriate, you might want to provide user instructions (such as the type of data that can be entered into a particular field) when you customize the screen.
- For multifile applications, wherever possible, display all other fields in a parent box before you display any join fields that link that box to a child box. In this way, all the fields of the parent box appear in one area and the user does not have to tab over a child box to complete an entry for the parent box.
- Display several records within a box where appropriate. When the primary purpose of an application is to display data, a box that displays more than one record is often more convenient.
- Provide a compressed format if many fields will appear in a box. A compressed format is one in which more than one field appears on a screen line. With a compressed format, the application is more likely to display an entire record on one screen page. This not only enhances the appearance of the screen, but also improves the usability of the application, since the user will not have to move to a new screen page to view the entire record.

- Wherever possible, try to limit the screen display to a single screen page. In particular, avoid splitting boxes across screens because users might update partial records if they cannot see all the fields of a record on a single screen page.

- Use a tabular screen format wherever appropriate. With this format, records appear to be organized in a table, a format that is familiar to most users.

- Exclude any fields from a box that do not need to appear. For some applications, a user does not need to see every field in a record. If a record contains particularly sensitive information, such as an employee's salary or age, consider excluding these fields from any application used solely for display purposes.

How To Customize a Screen

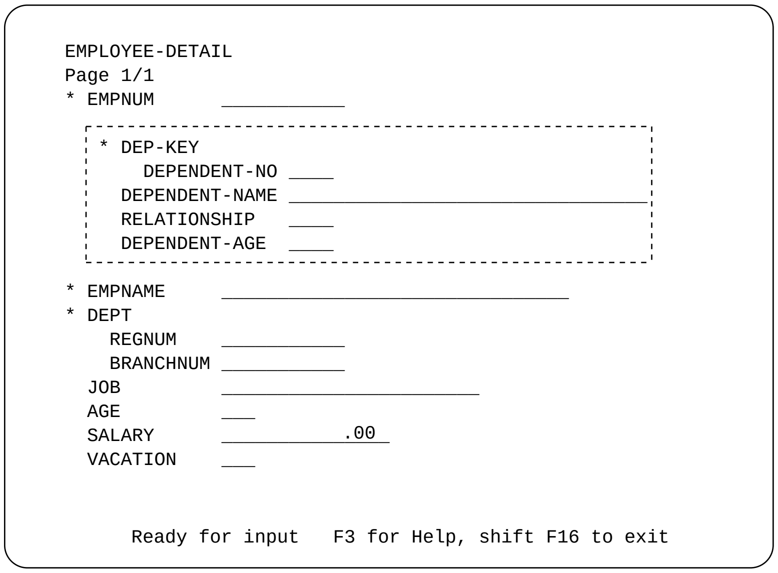
You can customize a screen for an application by supplying the appropriate values for the box and application attributes that determine screen format. Table 6-1 lists these attributes and indicates how a value supplied for each attribute affects the screen.

Table 6-1. Attributes That Affect Screen Format

Attribute Name	Effect of Supplying a Value
BOXTITLE 1 BOXTITLE 2 BOXTITLE 3	Supply a value for these attributes to provide user information that appears within a box on the screen or to provide your own screen labels for boxes with tabular formats.
EXCLUDE	Supply a value for this attribute to exclude fields from the screen.
HEADINGS	Supply a value for this attribute to indicate that DDL headings from the record description are to be used as screen labels or to indicate that ENABLE is not to supply screen labels for boxes with tabular formats.
INCLUDE	Supply a value for this attribute to define the order in which fields are to appear on the screen and, optionally, exclude fields from the screen.
SIZE	Supply a value for this attribute to display more than one record within a box.
SCREENFORMAT	Supply a value for this attribute to provide a compressed layout for a box.
TITLE	Supply a value for this attribute to provide your own screen title.
VALUES	Supply a value for this attribute to display initial values from the record description in screen fields.

Example of Screen Customization

The remainder of this discussion describes the tasks involved in providing a customized screen for "employee-info"--an application that displays information about employees and their dependents. This application accesses the same files ("employee" and "depend") as the application "employee-detail," described in Section 5. "Employee-detail" displays the standard screen shown in in Figure 6-1.

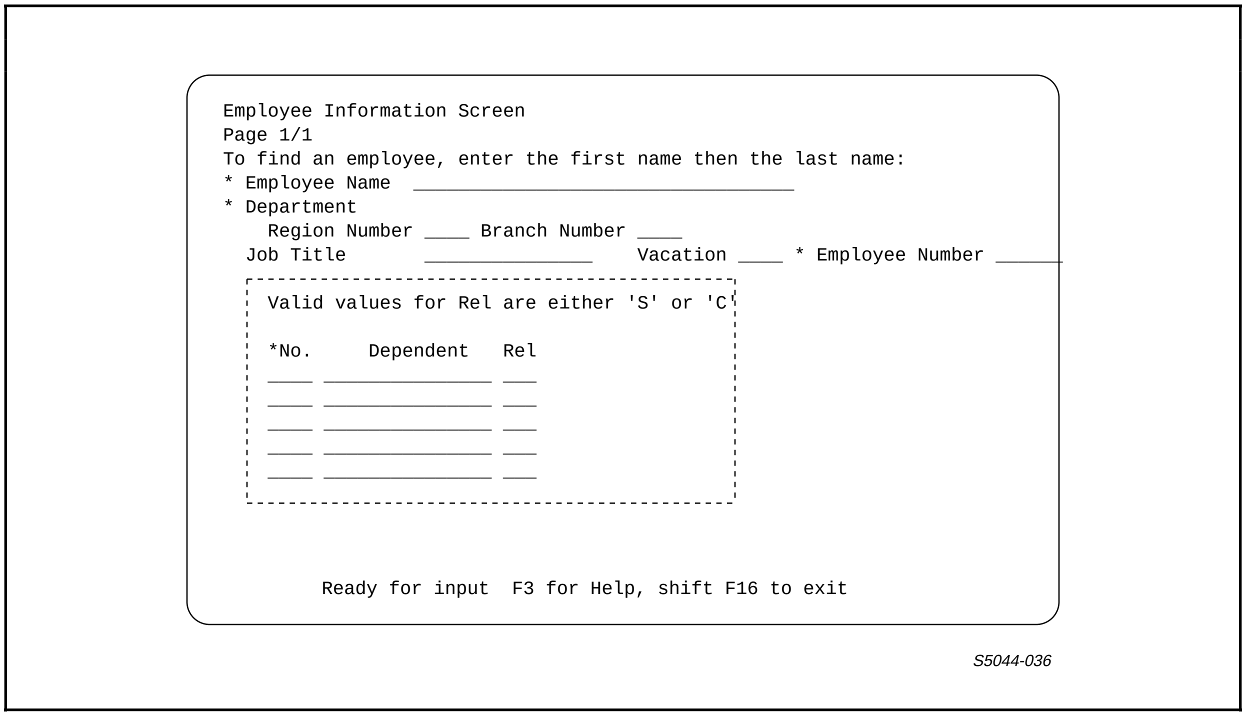


```
EMPLOYEE-DETAIL
Page 1/1
* EMPNUM      ___________

  * DEP-KEY
      DEPENDENT-NO ____
    DEPENDENT-NAME ________________________________
    RELATIONSHIP   ____
    DEPENDENT-AGE  ____

* EMPNAME     _______________________________
* DEPT
    REGNUM    ___________
    BRANCHNUM ___________
  JOB         _______________________
  AGE         ___
  SALARY      ___________.00
  VACATION    ___

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

S5044-035

Figure 6-1. Sample Standard Screen

For each step in the process, the discussion indicates an appropriate ENABLE command and illustrates the effect that each command has on screen appearance. Figure 6-2 shows the screen that is finally produced as a result of this customization.



```
Employee Information Screen
Page 1/1
To find an employee, enter the first name then the last name:
* Employee Name  __________________________________
* Department
    Region Number ____ Branch Number ____
  Job Title      _______________    Vacation ____ * Employee Number ______

    Valid values for Rel are either 'S' or 'C'

    *No.     Dependent   Rel
    ____ _______________ ___
    ____ _______________ ___
    ____ _______________ ___
    ____ _______________ ___
    ____ _______________ ___


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

Figure 6-2. Sample Customized Screen

To provide a customized screen for "employee-info," the following tasks are required:

1. Provide a descriptive screen title.

2. Indicate that DDL HEADINGS are to be used as screen labels for the box ("employ-box") that represents the "employee" file.

3. Supply a new order for the fields displayed in "employ-box." This new order will identify a key field, the employee name, as the first field that appears in "employ-box," and indicate that certain fields from the "employee" record description are not to appear on the screen.

4. Supply a compressed layout for "employ-box."

5. Supply instructions within "employ-box" that tell the user the correct way to enter an employee name.

6. Exclude a field ("dependent-age") that should not appear in the box ("depends-box") that represents the "depend" file.

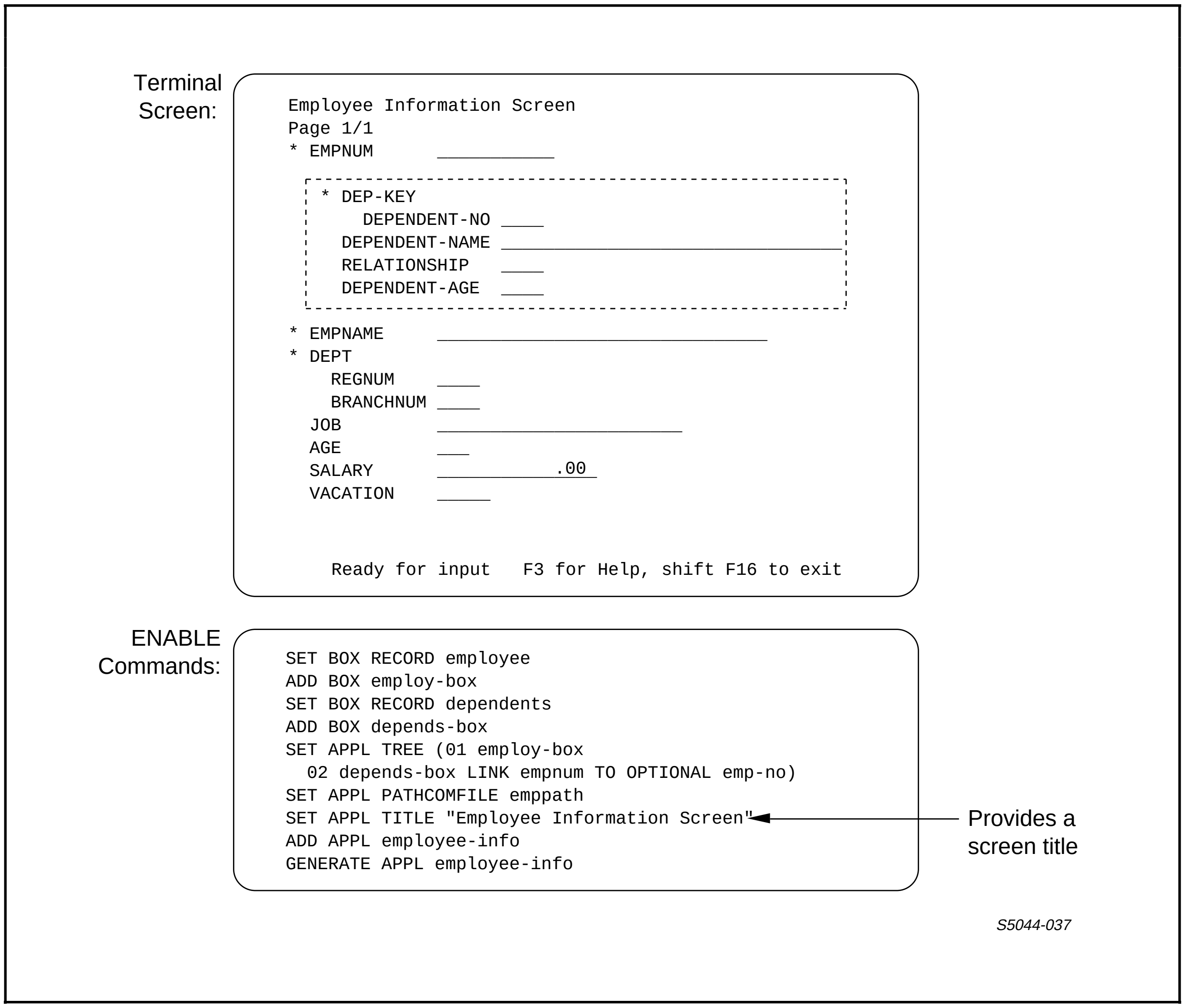
7. Provide a tabular format for "depends-box." With this format, the application can display up to five dependent records at one time. In addition, it supplies instructions that describe the data to be entered in this box.

Providing a Screen Title

Since the screen title is the first item that appears, you might begin customizing the screen by providing a descriptive screen title. To provide a descriptive screen title for "employee-info," for example, you could supply "Employee Information Screen" as a value for the TITLE attribute. To supply this value, you can include the following in the series of commands used to generate the application:

```
   SET APPL TITLE "Employee Information Screen"
```

Figure 6-3 shows the screen displayed by "employee-info" if you generate this application by supplying the attribute value described thus far. This figure also shows the SET TITLE command within the other commands used to generate the application.



```
Employee Information Screen
Page 1/1
* EMPNUM        ___________

  * DEP-KEY
       DEPENDENT-NO ____
     DEPENDENT-NAME ________________________________
     RELATIONSHIP   ____
     DEPENDENT-AGE  ____

* EMPNAME       _______________________________
* DEPT
    REGNUM      ____
    BRANCHNUM   ____
  JOB           _______________________
  AGE           ___
  SALARY        ___________.00
  VACATION      _____


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

```
SET BOX RECORD employee
ADD BOX employ-box
SET BOX RECORD dependents
ADD BOX depends-box
SET APPL TREE (01 employ-box
  02 depends-box LINK empnum TO OPTIONAL emp-no)
SET APPL PATHCOMFILE emppath
SET APPL TITLE "Employee Information Screen"
ADD APPL employee-info
GENERATE APPL employee-info
```

Figure 6-3. Screen With User-Defined Screen Title

If you supply a value for the TITLE attribute, the title can be up to 79 characters long (78 for T16-651x terminals). You can center the title in the following way:

1. Divide the number of characters in the title by 2.

2. Subtract the result of step 1 from 40 (character 40 falls at the center of the screen). This step gives you the number of blank characters that should precede the title.

To center the title "Employee Information Screen":

1. Divide 27 (the number of characters in the title) by 2 giving you a result of 13.

2. Subtract 13 from 40 giving you a result of 27.

3. Include 27 blank characters before you enter the title when you supply a value for the TITLE attribute:

   ```
   SET APPL TITLE "                           Employee &
   Information Screen"
   ```

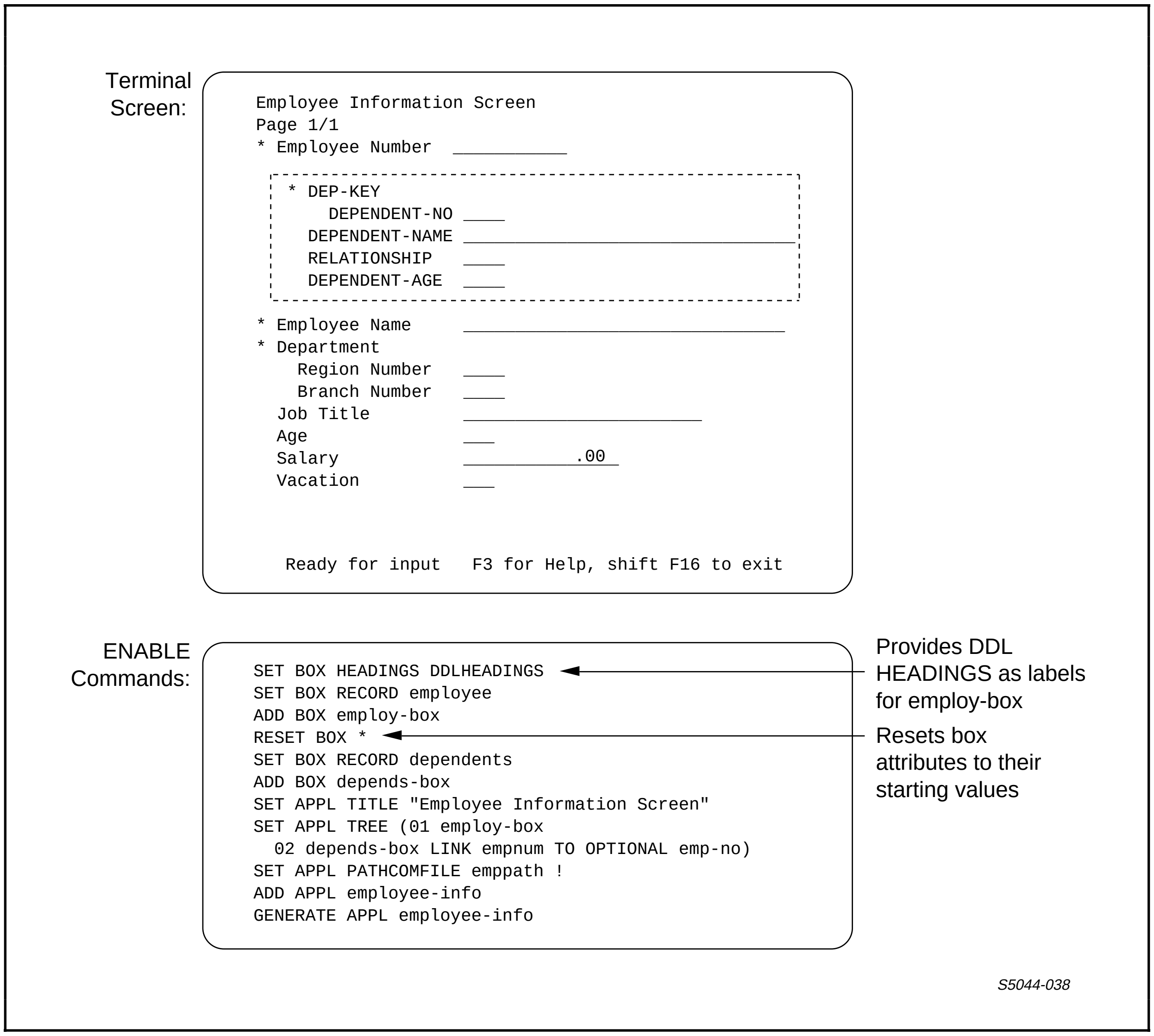
   Notice the ampersand character following the characters "Employee." This character indicates that the SET TITLE command is continued onto the next line. If you must continue your title text onto another line, be sure to include this character.

Using DDL Headings for Screen Labels

If the DDL record description associated with a file contains appropriate HEADING clauses, you can identify these headings as labels by supplying DDLHEADINGS as a value for the HEADING attribute. To use the DDL HEADING clauses from the "employee" record description for "employee-info," for example, include the following command when describing the "employ-box":

```
SET HEADINGS DDLHEADINGS
```

Figure 6-4 shows the screen that "employee-info" will display if you supply the attribute values discussed thus far. This figure also shows the SET HEADINGS command within the series of commands used to generate the application.


Terminal Screen:
ENABLE Commands:
Provides DDL HEADINGS as labels for employ-box
Resets box attributes to their starting values
S5044-038

```
Employee Information Screen
Page 1/1
* Employee Number  ___________

  * DEP-KEY
        DEPENDENT-NO ____
     DEPENDENT-NAME ________________________________
     RELATIONSHIP   ____
     DEPENDENT-AGE  ____

* Employee Name     _______________________________
* Department
    Region Number   ____
    Branch Number   ____
  Job Title         _______________________
  Age               ___
  Salary            ___________.00
  Vacation          ___


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

```
SET BOX HEADINGS DDLHEADINGS
SET BOX RECORD employee
ADD BOX employ-box
RESET BOX *
SET BOX RECORD dependents
ADD BOX depends-box
SET APPL TITLE "Employee Information Screen"
SET APPL TREE (01 employ-box
  02 depends-box LINK empnum TO OPTIONAL emp-no)
SET APPL PATHCOMFILE emppath !
ADD APPL employee-info
GENERATE APPL employee-info
```

Figure 6-4. Screen With DDL Headings as Screen Labels

A RESET command appears before the SET BOX command that identifies the "dependents" record description. When you use the SET command to supply values for the box attributes that describe one box, use the RESET command to reset these box attributes to their current values before you begin describing another box. If you use the SET BOX command but do not use the RESET command, ENABLE will use any current box attribute-values that remain in the attribute table for the next box that you add.

When you supply DDLHEADINGS as a value for the HEADINGS attribute, the application uses the field name as a label for any field in a record description that does not have a HEADING clause.

Reordering Screen Fields

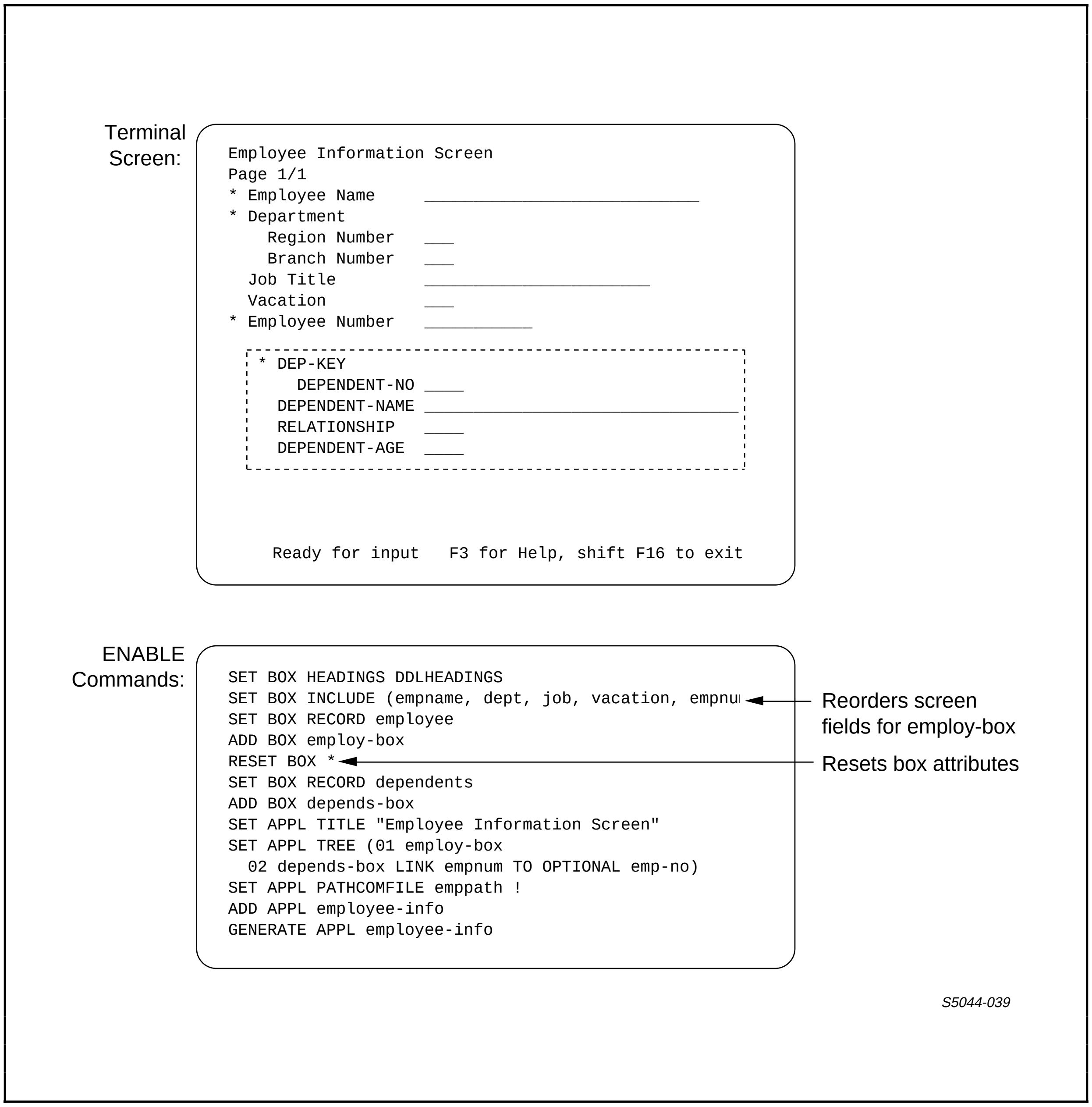
If you want to define the order in which fields appear on the screen, you must supply a value for the INCLUDE attribute. Consider, for example, the "employee-info" application. To provide an attractive and useful screen for this application, you should reorder the fields for "employ-box" so that:

- The join field ("empnum") appears after all the other fields in the box
- A key field (such as "empname") appears as the first field on the screen
- Fields (such as "age" and "salary") that contain sensitive information do not appear on the screen

You can reorder the fields in "employ-box" by including the following in the series of commands that describe this box:

```
   SET INCLUDE (empname, dept, job, vacation, empnum)
```

Figure 6-5 shows the screen that "employee-info" will display if you supply the attribute values discussed thus far. This figure also shows the preceding SET INCLUDE command in the series of commands used to generate the application.



```
Employee Information Screen
Page 1/1
* Employee Name     ____________________________
* Department
    Region Number   ___
    Branch Number   ___
  Job Title         _______________________
  Vacation          ___
* Employee Number   ___________

   * DEP-KEY
       DEPENDENT-NO ____
     DEPENDENT-NAME ________________________________
     RELATIONSHIP   ____
     DEPENDENT-AGE  ____



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

```
SET BOX HEADINGS DDLHEADINGS
SET BOX INCLUDE (empname, dept, job, vacation, empnu
SET BOX RECORD employee
ADD BOX employ-box
RESET BOX *
SET BOX RECORD dependents
ADD BOX depends-box
SET APPL TITLE "Employee Information Screen"
SET APPL TREE (01 employ-box
  02 depends-box LINK empnum TO OPTIONAL emp-no)
SET APPL PATHCOMFILE emppath !
ADD APPL employee-info
GENERATE APPL employee-info
```

Figure 6-5. Screen With Fields Reordered

When you supply a value for the INCLUDE attribute, consider the following:

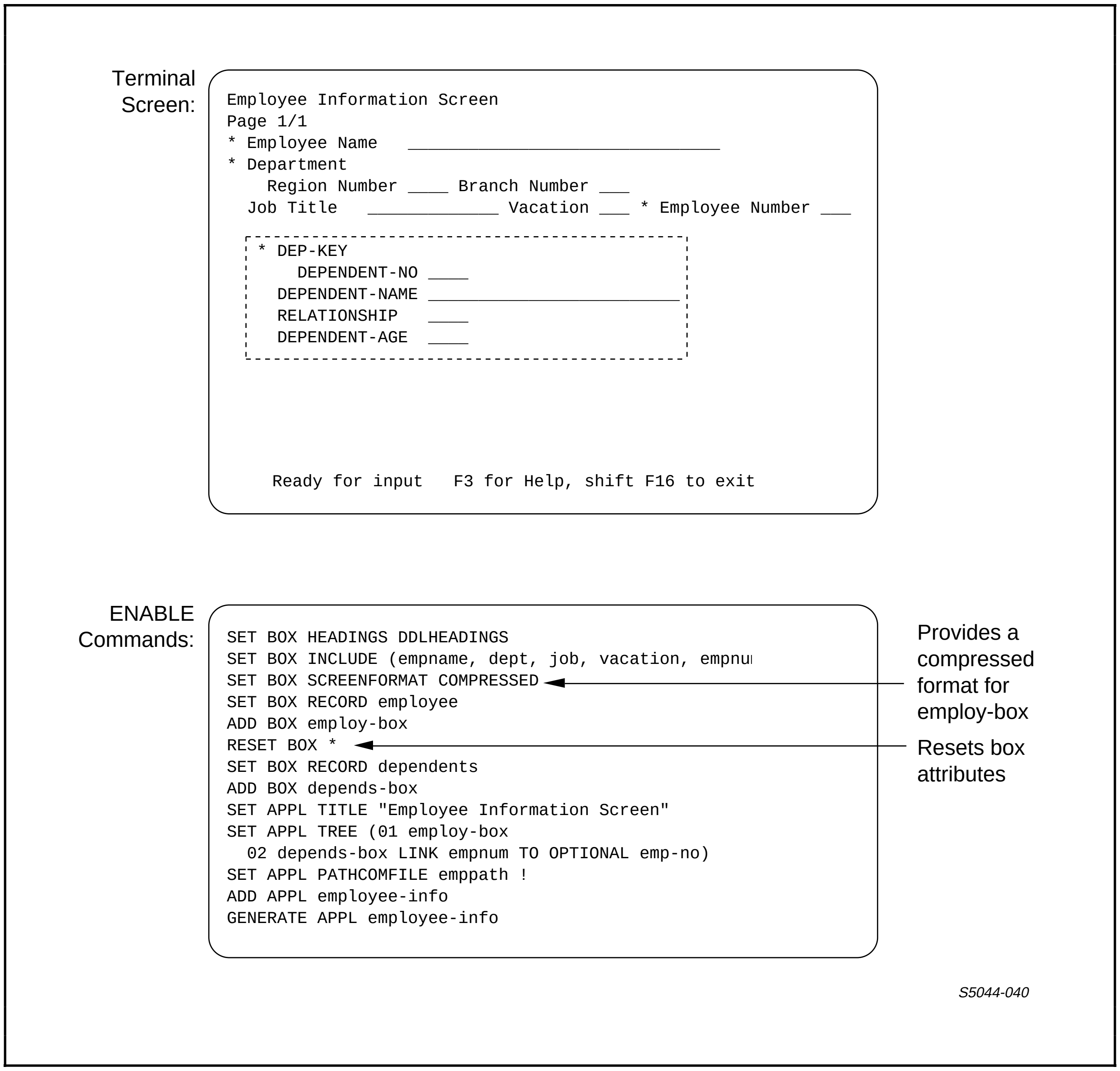
- If you do not include all the field names in a record description, the application does not display the omitted fields.
- If you use the application to update a record and one or more fields do not appear on the screen, the application uses the original values of the fields for the updated record.
- If you use the application to insert a record and one or more fields do not appear on the screen, the application uses a default value for these fields. The application uses zeros for numeric fields and blanks for alphanumeric or alphabetic fields.

Providing a Compressed Format

If several fields appear in a box, you might want to provide a compressed format for that box. A compressed format often improves the appearance of the screen. Consider, for example, the "employee-info" application. You could improve the appearance of the screen displayed by this application by supplying a compressed format for "employ-box." To provide this format, you could include the following in the series of commands used to describe "employ-box":

```
SET SCREENFORMAT COMPRESSED
```

Figure 6-6 shows the screen that "employee-info" will display if you supply the attribute values discussed thus far. This figure also shows the preceding SET SCREENFORMAT command in the series of commands used to generate the application.



```
Employee Information Screen
Page 1/1
* Employee Name   _______________________________
* Department
    Region Number ____ Branch Number ___
  Job Title   _____________ Vacation ___ * Employee Number ___

   * DEP-KEY
       DEPENDENT-NO ____
     DEPENDENT-NAME _________________________
     RELATIONSHIP   ____
     DEPENDENT-AGE  ____

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

```
SET BOX HEADINGS DDLHEADINGS
SET BOX INCLUDE (empname, dept, job, vacation, empnu
SET BOX SCREENFORMAT COMPRESSED
SET BOX RECORD employee
ADD BOX employ-box
RESET BOX *
SET BOX RECORD dependents
ADD BOX depends-box
SET APPL TITLE "Employee Information Screen"
SET APPL TREE (01 employ-box
  02 depends-box LINK empnum TO OPTIONAL emp-no)
SET APPL PATHCOMFILE emppath !
ADD APPL employee-info
GENERATE APPL employee-info
```

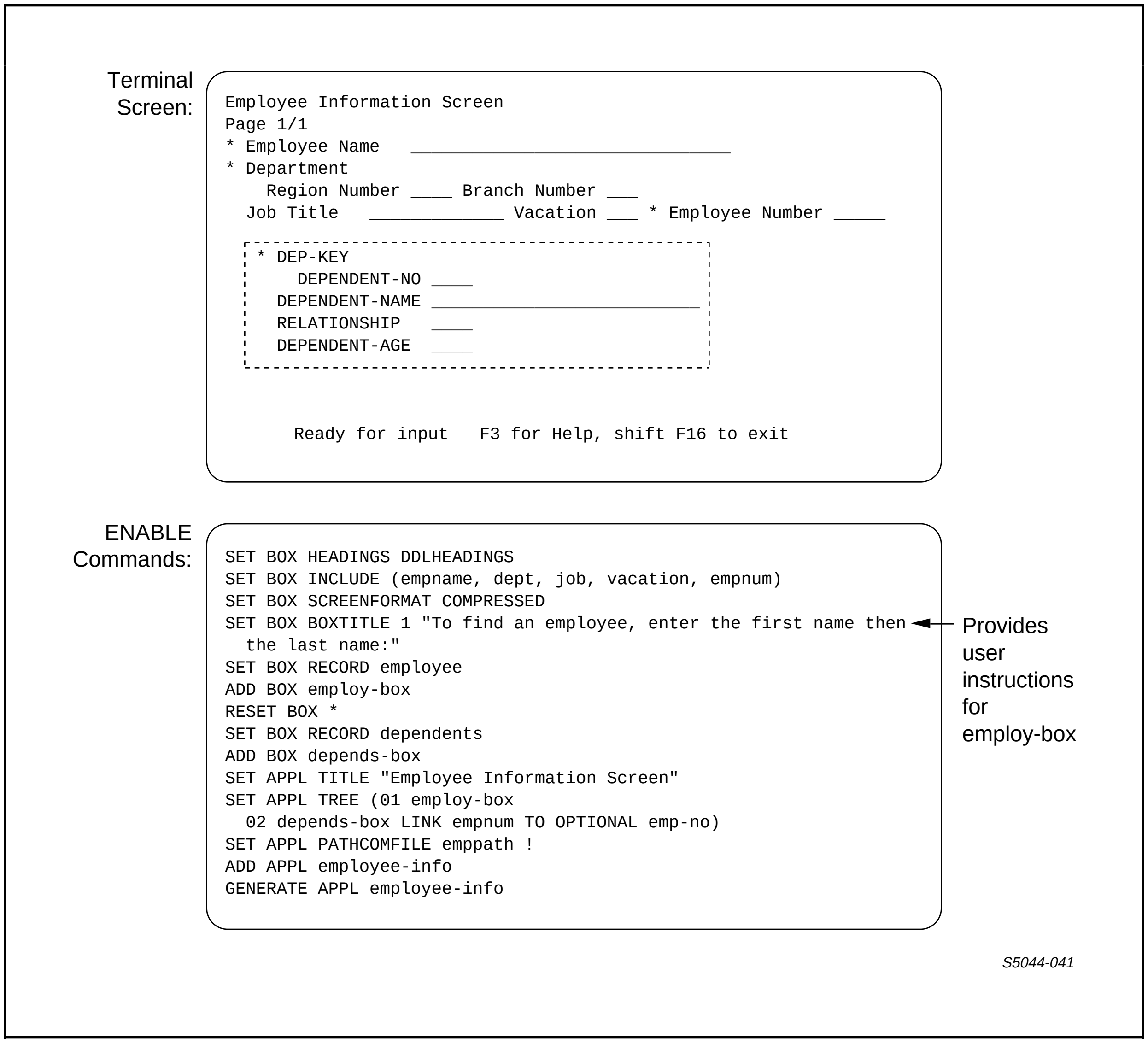
Figure 6-6. Screen With Compressed Format

Providing User Information for a Box

For a variety of reasons, you might want to provide user information to be displayed within a box. Consider, for example, the "employee-info" application. A user of this application might not know how to enter an employee name. You can provide the user with this information by supplying a value for one of the BOXTITLE attributes when you describe "employ-box," for example:

```
   SET BOXTITLE 1 "To find an employee, enter the first name &
   then the last name:"
```

Figure 6-7 shows the screen that "employee-info" will display if you supply the attribute values discussed thus far. This figure also shows the preceding SET BOXTITLE 1 command in the series of commands used to generate the application.



```
Employee Information Screen
Page 1/1
* Employee Name   _______________________________
* Department
    Region Number ____ Branch Number ___
  Job Title    _____________ Vacation ___ * Employee Number _____

   * DEP-KEY
       DEPENDENT-NO ____
     DEPENDENT-NAME __________________________
     RELATIONSHIP   ____
     DEPENDENT-AGE  ____


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

```
SET BOX HEADINGS DDLHEADINGS
SET BOX INCLUDE (empname, dept, job, vacation, empnum)
SET BOX SCREENFORMAT COMPRESSED
SET BOX BOXTITLE 1 "To find an employee, enter the first name then
  the last name:"
SET BOX RECORD employee
ADD BOX employ-box
RESET BOX *
SET BOX RECORD dependents
ADD BOX depends-box
SET APPL TITLE "Employee Information Screen"
SET APPL TREE (01 employ-box
  02 depends-box LINK empnum TO OPTIONAL emp-no)
SET APPL PATHCOMFILE emppath !
ADD APPL employee-info
GENERATE APPL employee-info
```

Figure 6-7. Screen With User-Defined Information Displayed in a Box

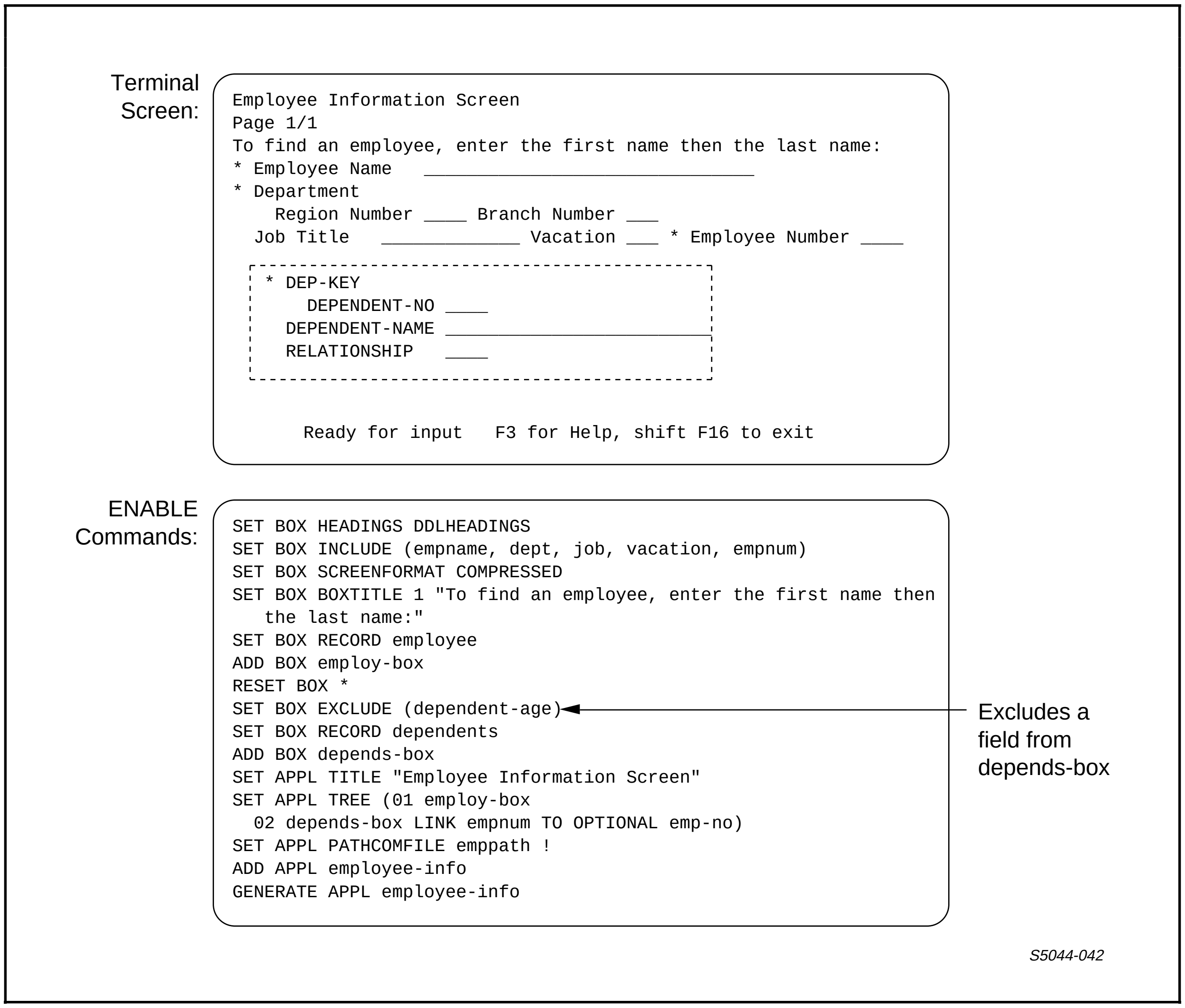
You can provide up to three lines of user information within a box by supplying values for the BOXTITLE 1, BOXTITLE 2, and BOXTITLE 3 attributes.

Excluding Fields From a Box

If the users of an application do not need to see the contents of certain fields, you can exclude these fields from display on the screen. Consider, for example, the "employee-info" application. Since this application is to be used to display general information about employees and their dependents, you might not want to display a dependent's age. You could exclude the field ("dependent-age") that stores a dependent's age by specifying the following in the series of commands that describe "depends-box":

```
   SET EXCLUDE (dependent-age)
```

Figure 6-8 shows the screen that "employee-info" will display if you supply the attribute values discussed thus far. This figure also shows the preceding SET EXCLUDE command within the series of commands used to generate the application.



```
Employee Information Screen
Page 1/1
To find an employee, enter the first name then the last name:
* Employee Name   _______________________________
* Department
    Region Number ____ Branch Number ___
  Job Title    _____________ Vacation ___ * Employee Number ____

   * DEP-KEY
       DEPENDENT-NO ____
     DEPENDENT-NAME _________________________
     RELATIONSHIP   ____


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

```
SET BOX HEADINGS DDLHEADINGS
SET BOX INCLUDE (empname, dept, job, vacation, empnum)
SET BOX SCREENFORMAT COMPRESSED
SET BOX BOXTITLE 1 "To find an employee, enter the first name then
   the last name:"
SET BOX RECORD employee
ADD BOX employ-box
RESET BOX *
SET BOX EXCLUDE (dependent-age)
SET BOX RECORD dependents
ADD BOX depends-box
SET APPL TITLE "Employee Information Screen"
SET APPL TREE (01 employ-box
  02 depends-box LINK empnum TO OPTIONAL emp-no)
SET APPL PATHCOMFILE emppath !
ADD APPL employee-info
GENERATE APPL employee-info
```

Figure 6-8. Screen With Field Excluded

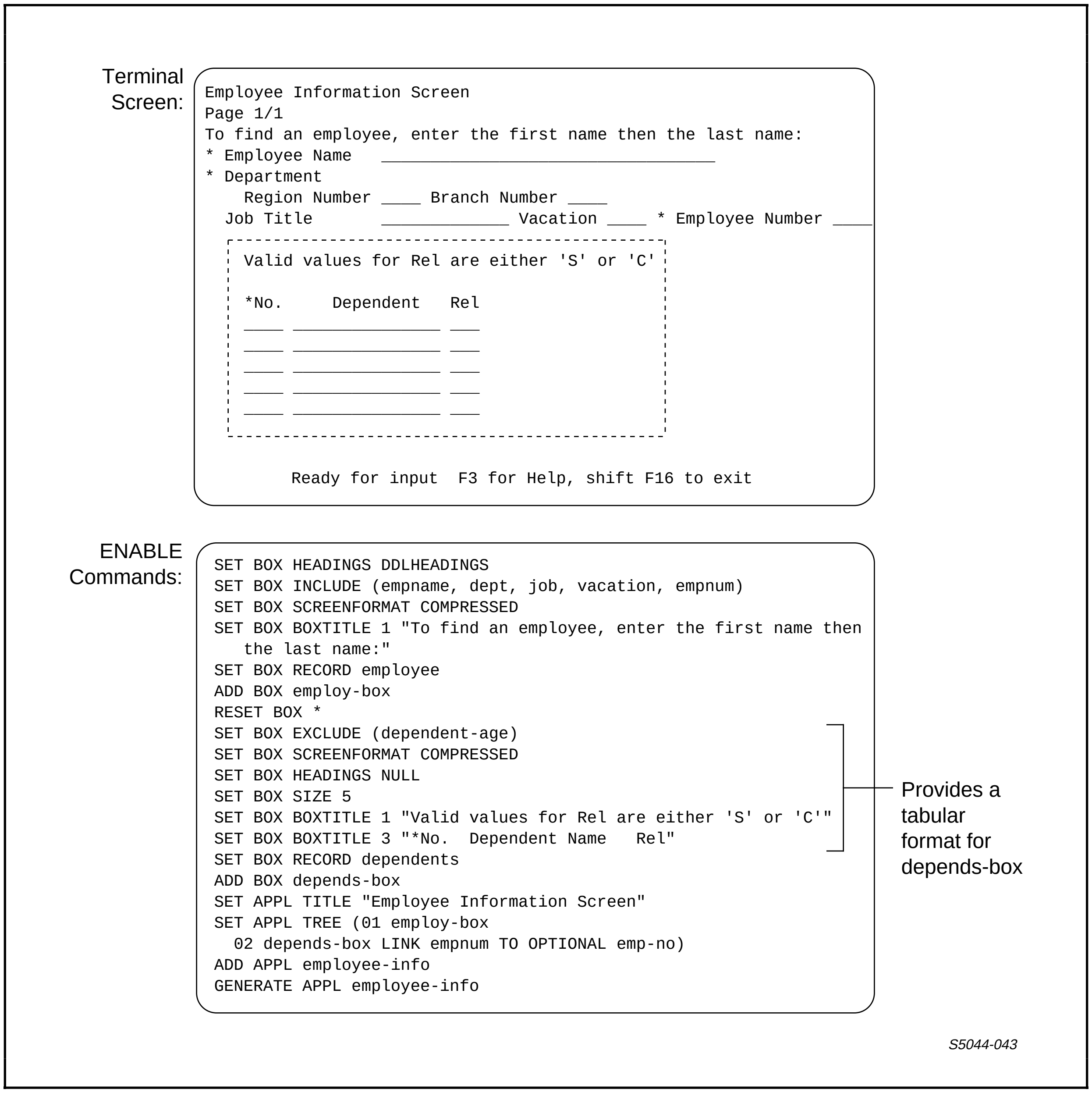
Providing a Tabular Format

If you want to display several records within a box, you can enhance the appearance of the screen by providing a tabular format for the box. Consider, for example, the "employee-info" application. Since one employee record can match (have the same join-field value) several dependent records, you might want to provide the ability to display several dependent records at one time. You can specify a tabular format for "depends-box" (the box that represents the dependent records) by including the following commands that describe this box:

```
   SET BOX SCREENFORMAT COMPRESSED
   SET BOX HEADINGS NULL
   SET BOX SIZE 5
   SET BOX BOXTITLE 1 "Valid values for Rel are either S or C."
   SET BOX BOXTITLE 3 "*No.  Dependent Name     Rel"
```

Notice that the value of the BOXTITLE 1 attribute supplies user information while the value of the BOXTITLE 3 attribute supplies screen labels for "depends-box."

Figure 6-9 shows the final customized screen displayed by "employee-info." This figure also shows the preceding ENABLE commands within the series of commands used to generate the application.



```
Employee Information Screen
Page 1/1
To find an employee, enter the first name then the last name:
* Employee Name   __________________________________
* Department
    Region Number ____ Branch Number ____
  Job Title       _____________ Vacation ____ * Employee Number ____

   Valid values for Rel are either 'S' or 'C'

   *No.     Dependent   Rel
   ____ _______________ ___
   ____ _______________ ___
   ____ _______________ ___
   ____ _______________ ___
   ____ _______________ ___


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

```
SET BOX HEADINGS DDLHEADINGS
SET BOX INCLUDE (empname, dept, job, vacation, empnum)
SET BOX SCREENFORMAT COMPRESSED
SET BOX BOXTITLE 1 "To find an employee, enter the first name then
   the last name:"
SET BOX RECORD employee
ADD BOX employ-box
RESET BOX *
SET BOX EXCLUDE (dependent-age)
SET BOX SCREENFORMAT COMPRESSED
SET BOX HEADINGS NULL
SET BOX SIZE 5
SET BOX BOXTITLE 1 "Valid values for Rel are either 'S' or 'C'"
SET BOX BOXTITLE 3 "*No.  Dependent Name   Rel"
SET BOX RECORD dependents
ADD BOX depends-box
SET APPL TITLE "Employee Information Screen"
SET APPL TREE (01 employ-box
  02 depends-box LINK empnum TO OPTIONAL emp-no)
ADD APPL employee-info
GENERATE APPL employee-info
```

Figure 6-9. Screen With Tabular-Format Box

Determining if a Record Will Fit in a Tabular Format

You can provide a tabular format for a box as long as all the fields displayed for one record will fit in the box. Up to 71 characters will fit in a box at the second level of the tree structure. At each subsequent level, subtract 8 characters to determine the number of characters that will fit.

When you determine whether a particular record will fit within a tabular format, allow for the following:

- The blank character that an application displays before and after a record

- The blank character (or characters) that an application displays between fields

  --For T16-652x, T16-653x, and IBM-327x terminals, an application displays one blank character between each field

  --For T16-651x terminals, an application displays two blank characters between each field

Positioning Labels Over Screen Fields

When you use the BOXTITLE attributes to provide screen labels, you want these labels to appear over the appropriate screen fields. To make sure that the labels appear in the correct position, enter a comment line that shows the way the fields will appear in the box immediately before the SET BOXTITLE command; for example:

```
...
--              ## #################### # --
SET BOXTITLE 3 "
```

In the preceding example, the pound sign symbols (#) indicate where fields will appear within a box. If you place a similar comment line in an edit file with your ENABLE commands, you can use the comment line to center your labels in the SET BOXTITLE command.

DEFINING THE OPERATIONS THAT AN APPLICATION CAN PERFORM

A basic ENABLE application can perform delete, insert, read, or update operations on the data base files to which it has access. Depending on the purpose for which you are generating the application, you might want either to limit or to enhance these operations for a particular application.

To determine the kind of operations that you want to allow, ask yourself the following questions about each box used by the application:

- Is the purpose of the application to display information in this box? If so, consider limiting the application to read-only operations for the box.

- Will a user of the application need to delete or update existing records in the box? If not, consider limiting the application to read and insert operations only for the box.

- Is the box a child box? If so, consider providing the application with the ability to perform an automatic read operation for the box. Although providing this ability might slow processing time slightly, it greatly enhances the usability of the application since the user does not have to press a function key to request the read operation.

Define the operations that an application can perform by supplying values for the ABILITY (or file operation) attributes. These attributes are: DELETE, FILL, INSERT, READ, and UPDATE.

Limiting the Operations an Application Can Perform

To limit the operations that an application can perform on a particular file, supply OFF as a value for the appropriate ABILITY attribute when you describe the box that represents the file. Consider, for example, the "employee-info" application described earlier in this section. Since the purpose of this application is to display information about employees and their dependents, "employee-info" only requires read access to the "employee" and "depend" files. You can restrict the application to read-only operations by using the following SET command before you add "employ-box" and "depends-box":

```
   SET BOX DELETE OFF, INSERT OFF, UPDATE OFF
```

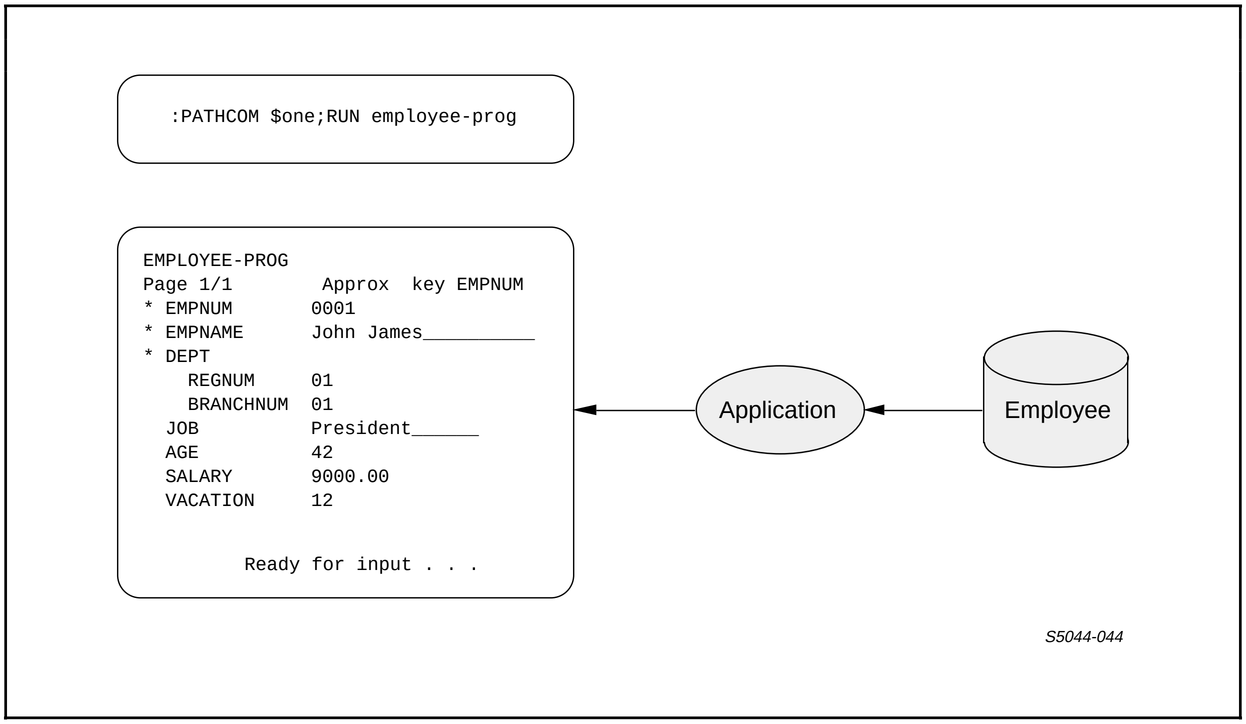
Providing Automatic READ Operations

By supplying ON as a value for the FILL attribute, you generate an application that can perform an automatic read operation under certain conditions. To determine when the application should perform this automatic read operation, ENABLE examines the level at which a box with FILL ON resides in the tree structure.

Automatic READ Operations for a Parent Box or the Only Box in the Tree

If you supply ON for the FILL attribute of a parent box or the only box in the tree structure, the application, upon execution, will perform an automatic read operation upon the file represented by that box.

Suppose, for example, that you generate the application ("employee-prog") described in Section 4 with FILL ON. Since this application uses only one box ("employee"), the application will, upon execution, read the first record in the "employee" file, as illustrated in Figure 6-10.



```
:PATHCOM $one;RUN employee-prog
```

```
EMPLOYEE-PROG
Page 1/1         Approx  key EMPNUM
* EMPNUM        0001
* EMPNAME       John James__________
* DEPT
    REGNUM      01
    BRANCHNUM   01
  JOB           President______
  AGE           42
  SALARY        9000.00
  VACATION      12

         Ready for input . . .
```

Figure 6-10. Automatic READ for First Level Box

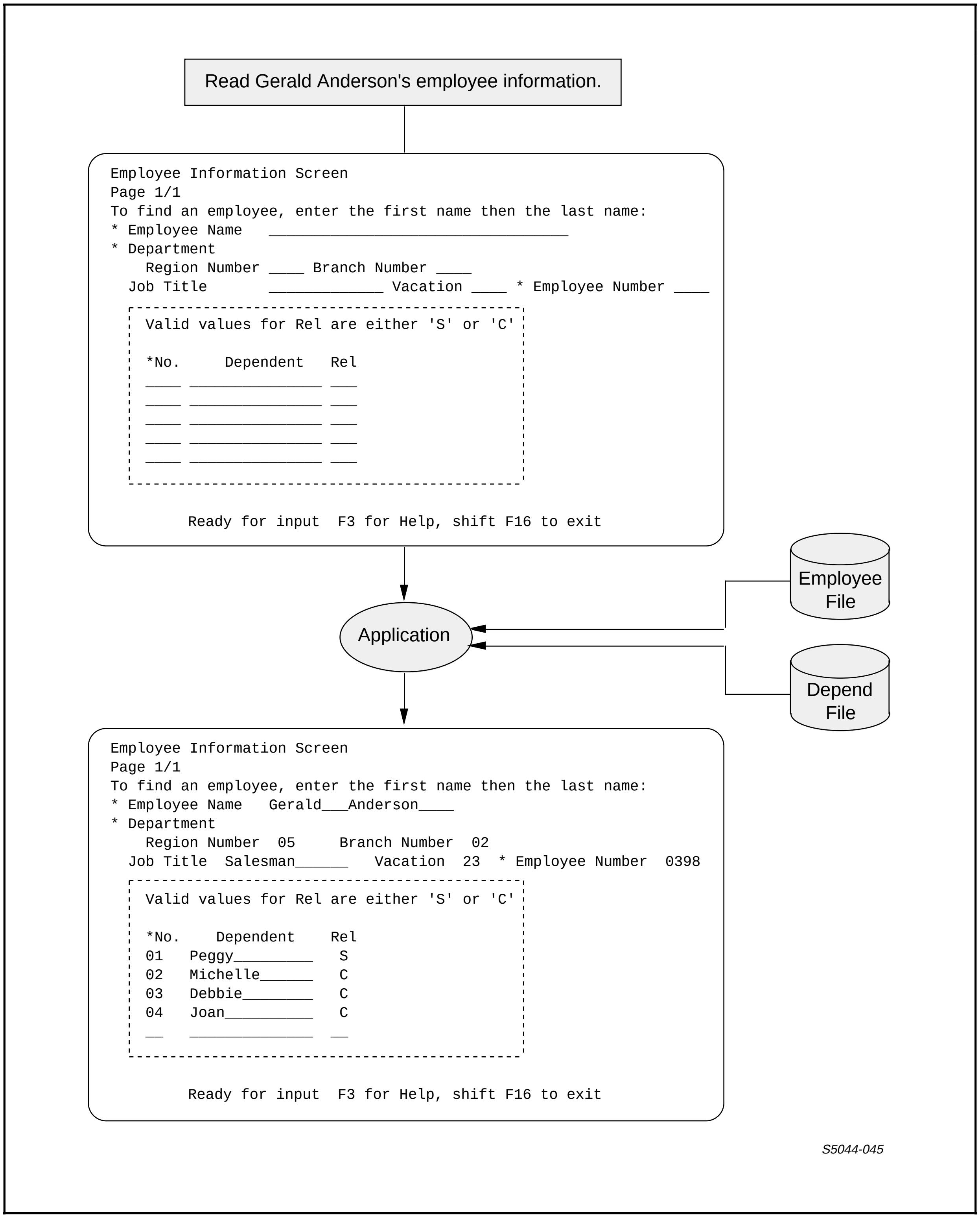
If you supply ON as a value for both FILL and VALUES attributes for a box, the application will display initial values from a record description, even if no records exist in a file. In some cases, appearance of these initial values might be confusing to the person who uses the application. In other cases the initial may be an enhancement. These cases are described later in this section.

Automatic READ Operations for a Child Box

If you supply ON for the FILL attribute of a child box, an application performs an automatic read for the child box whenever an operation changes values in the parent box. Consider, for example, the application ("employee-info") described earlier in this section. You can generate this application so that it automatically reads records for "depends-box" whenever a user performs an operation on "employ-box." To provide this ability for "employee-info," you must generate the application with FILL ON for "depends-box" as in the following series of commands:

```
   SET HEADINGS DDLHEADINGS
   SET BOX INCLUDE (empname, dept, job, vacation, empnum)
   SET BOX SCREENFORMAT COMPRESSED
   SET BOXTITLE 1 "To find an employee, enter the first name then
   the& last name:"
   SET BOX RECORD employee
   ADD BOX employ-box
   RESET BOX *

   SET BOX SCREENFORMAT COMPRESSED
   SET BOX HEADINGS NULL
   SET BOX SIZE 5
   SET BOX BOXTITLE 1 "Valid values for Rel are either 'S' or &
   'C'"
   SET BOX BOXTITLE 3 "*No.  Dependent Name     Rel"
   SET BOX FILL ON  --  Requests automatic read operations  --
   SET BOX RECORD dependents
   ADD BOX depends-box
   SET APPL TITLE "Employee Information Screen"
   SET APPL TREE (01 employ-box
                     02 depends-box LINK empnum
                     TO OPTIONAL emp-no)
   SET APPL PATHCOMFILE emppath !
   ADD APPL employee-info
   GENERATE employee-info
```

Figure 6-11 shows the events that occur when an application, generated using the above commands, reads an "employee" record.



```
Employee Information Screen
Page 1/1
To find an employee, enter the first name then the last name:
* Employee Name   __________________________________
* Department
    Region Number ____ Branch Number ____
  Job Title       _____________ Vacation ____ * Employee Number ____

    Valid values for Rel are either 'S' or 'C'

    *No.     Dependent   Rel
    ____ _______________ ___
    ____ _______________ ___
    ____ _______________ ___
    ____ _______________ ___
    ____ _______________ ___

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

```
Employee Information Screen
Page 1/1
To find an employee, enter the first name then the last name:
* Employee Name   Gerald___Anderson____
* Department
    Region Number  05     Branch Number  02
  Job Title  Salesman______   Vacation  23  * Employee Number  0398

    Valid values for Rel are either 'S' or 'C'

    *No.    Dependent    Rel
    01   Peggy_________   S
    02   Michelle______   C
    03   Debbie________   C
    04   Joan__________   C
    __   ______________  __

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

Figure 6-11. FILL ON for a Second Level Box

Restricting an Application to a Subset of Records for a Child Box

You can restrict an application to a subset of records for a child box by allowing it to access only a single record for the parent box. Suppose, for example, that you want to generate an application that can access the employee records of only those employees who belong to region 1 and branch 2. If this region and branch information is also stored in a file that contains general departmental information, you can make a box that represents these departmental records the parent of the employee records. You can then restrict the application to a subset of the employee records by:

1. Including a VALUE clause in the DDL record description of the departmental records. To limit the application to the subset of employee records where the employees belong to region 1 and branch 2, the VALUE clauses in this record description might appear as follows:

```
RECORD department-limit.
FILE IS dept KEY-SEQUENCED.
  02 dept-num.
    04 regnum        PIC 99    VALUE 01.
    04 branchnum     PIC 99    VALUE 02.
  02 department-name PIC X(18) VALUE "California South ".
KEY 0 IS dept-num.
KEY "dn" IS dept-name.
END
```

   If the "dept" file already had a record description that did not contain the appropriate VALUE clauses, you could add a new record description for this file. When you add this new record description to the dictionary, it is good practice to use a different record name. (ENABLE can use more than one record description per file, so long as the record lengths are the same for both.)

2. Supplying ON as a value for the FILL and VALUES attributes and OFF as a value for the DELETE, INSERT, READ, and UPDATE attributes when you add the box that represents the "dept" file, for example:

```
SET BOX RECORD department-limits
SET BOX VALUES ON, FILL ON
SET BOX DELETE OFF, INSERT OFF, READ OFF, UPDATE OFF
ADD BOX department-box
```

When you supply ON as a value for the FILL and VALUES attributes, the application, upon execution, reads the record in the "dept" file whose field values match the field values described in the DDL VALUE clause. Since you have supplied OFF for the values of the DELETE, INSERT, READ, and UPDATE attributes, the application cannot perform any other file operation on the "department-box." Therefore, the application cannot retrieve any other record for this box.

3. Resetting the box attributes before you add a box to represent the "employee" file, as follows:

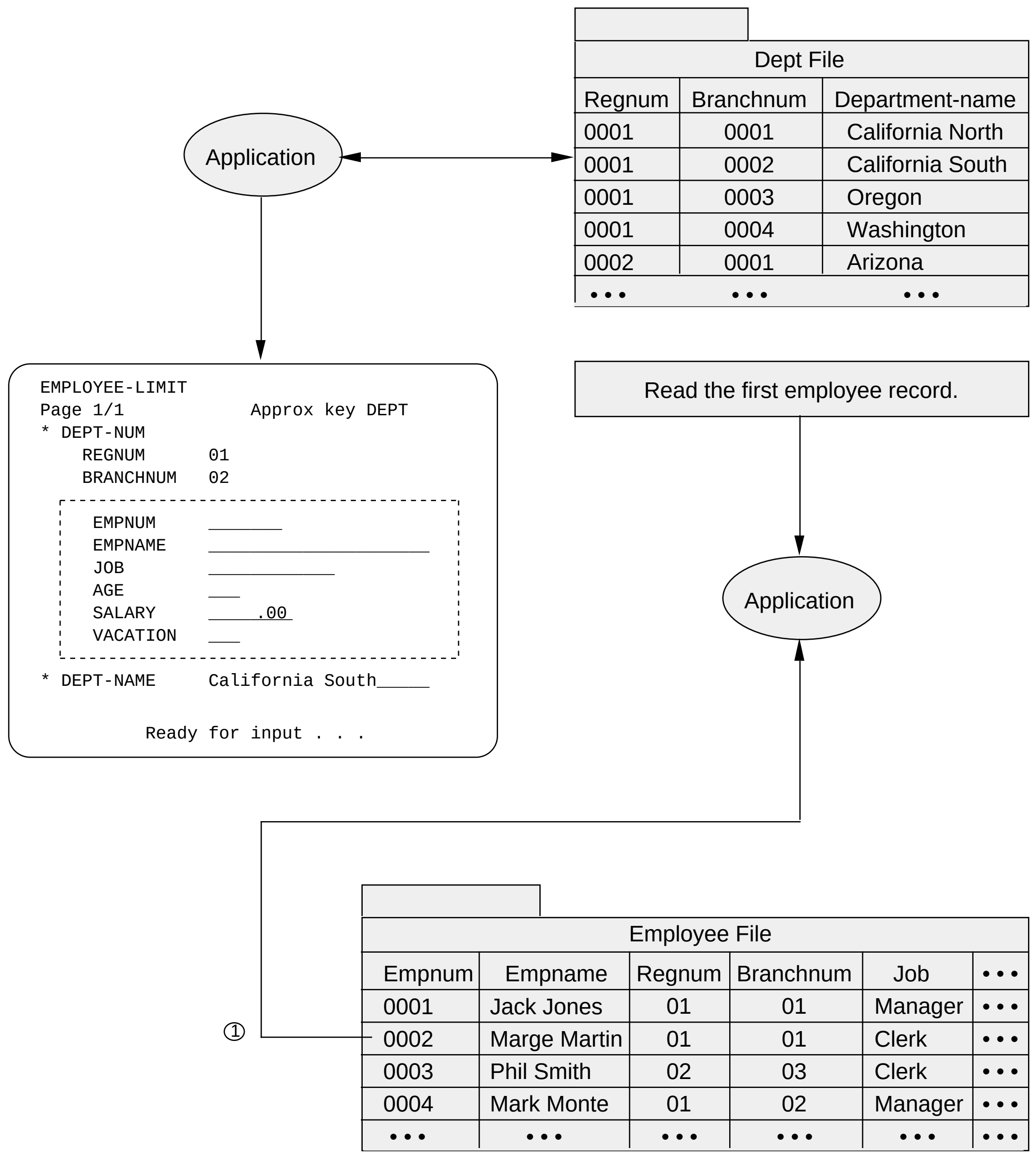
```
RESET BOX *
SET BOX RECORD employee
ADD BOX employ-box
```

   If you do not include the RESET command, ENABLE will use the current value (OFF) of the DELETE, INSERT, READ, and UPDATE attributes for "employ-box." This means that the application could not perform any file operation on this box.

Because the application can only access a single department-box record, it can only read or insert records in employ-box that match that single department-box record.

Figure 6-12 illustrates the events that occur when you execute an application generated with the following ENABLE commands:

```
SET BOX RECORD department-limits
SET BOX VALUES ON, FILL ON
SET BOX DELETE OFF, INSERT OFF, READ OFF, UPDATE OFF
ADD BOX department-box
RESET BOX *
SET BOX RECORD employee
ADD BOX employ-box
SET APPL TREE (01 department-box
                 02 employ-box LINK dept-num TO OPTIONAL dept)
SET APPL PATHCOMFILE employ-limit
ADD APPL employ-limit
GENERATE APPL employ-limit
```



Dept File

Regnum	Branchnum	Department-name
0001	0001	California North
0001	0002	California South
0001	0003	Oregon
0001	0004	Washington
0002	0001	Arizona
• • •	• • •	• • •

```
EMPLOYEE-LIMIT
Page 1/1            Approx key DEPT
* DEPT-NUM
    REGNUM      01
    BRANCHNUM   02

     EMPNUM     _______
     EMPNAME    _____________________
     JOB        ____________
     AGE        ___
     SALARY     _____.00
     VACATION   ___

* DEPT-NAME     California South_____


          Ready for input . . .
```

Employee File

Empnum	Empname	Regnum	Branchnum	Job	• • •
0001	Jack Jones	01	01	Manager	• • •
0002	Marge Martin	01	01	Clerk	• • •
0003	Phil Smith	02	03	Clerk	• • •
0004	Mark Monte	01	02	Manager	• • •
• • •	• • •	• • •	• • •	• • •	• • •

Legend

① The application can only read an employee record if Regnum = 01 and Branchnum = 02.

S5044-046

Figure 6-12. Application Limited to Subset of Records

ENSURING FILE INTEGRITY

Because the starting value of the CHECKDATA attribute is ON, a basic ENABLE application ensures the integrity of your data base files by not allowing a user to enter invalid numeric data in these files. In this case, the term invalid data refers to data that is of the wrong data type. You can provide one of two additional methods of ensuring the integrity of your data base files by supplying values for either the TMF or NONSTOP attributes.

If a data base file is audited by the Transaction Monitoring Facility (TMF), you can supply ON as a value for the TMF attribute when you describe the box that represents this file. If you supply ON for the TMF attribute, ENABLE includes the special code required by TMF in the portion of the SCREEN COBOL requester program that refers to that box.

If a file is not audited by TMF, you can supply ON as a value for the NONSTOP attribute. When NONSTOP is ON for a box, the copy of the General Server that is used for the box runs as a NonStop process pair. This means that a backup copy of the General Server process is always available to take over if the primary process fails.

In addition, you can generate an application that accesses both audited files and nonaudited files. In this case, you must supply a different copy of the General Server for each group of files.

Identifying Files Audited by TMF

If an application is to maintain (perform DELETE, INSERT, or UPDATE operations) a file audited by TMF, the value of the TMF attribute for that box must be ON. For example, the following series of commands supplies TMF ON as a value for "employ-box":

```
   SET BOX TMF ON
   SET BOX RECORD employee
   ADD BOX employ-box
```

If the purpose of an application is to display information from a file audited by TMF, you do not have to set TMF ON for the box that represents this file. In this case, however, you must supply OFF as the value of the DELETE, INSERT, and UPDATE attributes.

Running the General Server as a NonStop Process Pair

If a file is not audited by TMF, you can indicate that the General Server is to run as a NonStop process pair when accessing that file. The following series of commands supplies NONSTOP ON for "depends-box":

```
   SET BOX NONSTOP ON
   SET BOX RECORD dependents
   ADD BOX depends-box
```

Accessing Audited and Nonaudited Files With the Same Application

You can generate an application that can access both audited and nonaudited files. If you want to generate such an application, you must define at least two server classes for the General Server. You define a server class by supplying a value for the SERVERCLASS attribute.

Suppose, for example, that the following is true about the files to be accessed by "employee-info" (the application described earlier in this section):

- "Employee" is audited by TMF.
- "Depend" is not audited by TMF.

To supply TMF ON for "employ-box" (the box that represents the "employee" file) supply a server class name for the copy of the General Server that will access this file. You could supply a different server class name by including the following in the series of commands used to generate "employee-info":

```
   ...

   SET BOX TMF ON  <---------------  indicates "employ-box" is
                                     audited by TMF
   SET BOX SERVERCLASS tmf-serv <--  provides a server class name
                                     for the copy of the General
                                     Server used for "employ-box"
   SET BOX RECORD employee
   ADD BOX employ-box
   RESET BOX TMF, SERVERCLASS  <---  resets TMF and SERVERCLASS
                                     to their starting values,
                                     OFF and ENABLE-SERVER
                                     respectively
   SET BOX RECORD dependents
   ADD BOX depends-box
```

Figure 6-13 illustrates the "employee-info" application during execution. Note that a copy of the General Server named "tmf-serv" accesses the "employee" file while a copy of the General Server named ENABLE-SERVER accesses the "depend" file.

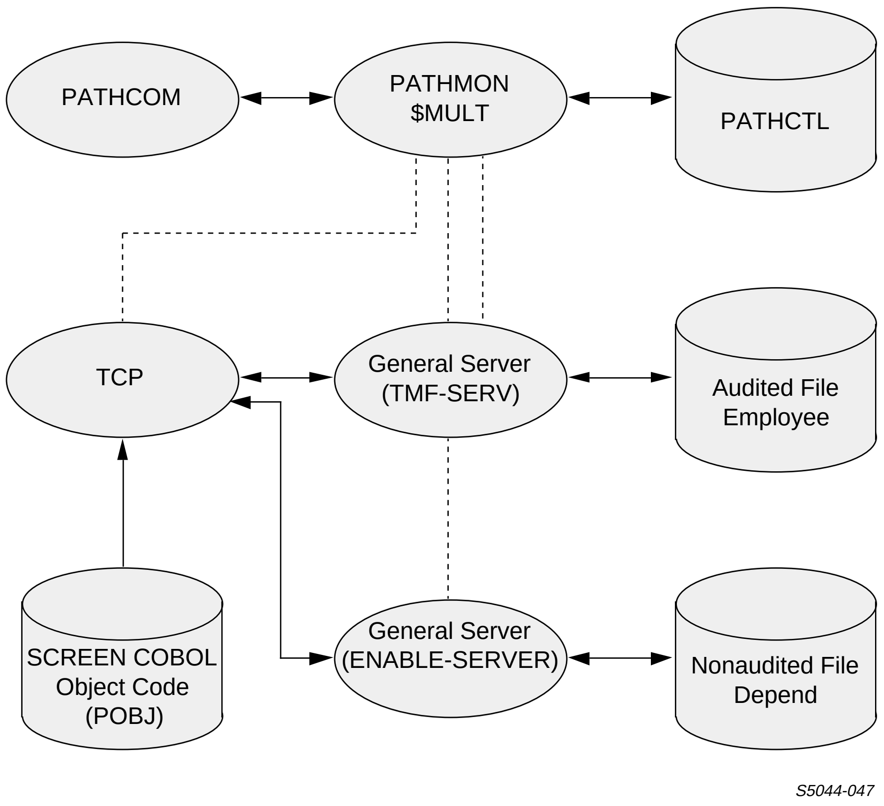

PATHCOM
PATHMON $MULT
PATHCTL
TCP
General Server (TMF-SERV)
Audited File Employee
SCREEN COBOL Object Code (POBJ)
General Server (ENABLE-SERVER)
Nonaudited File Depend
S5044-047

Figure 6-13. Accessing Audited and Nonaudited Files

SECTION 7

MODIFYING A GENERATED APPLICATION

You can tailor an application further by modifying the SCREEN COBOL source code. To modify the source code, you must:

1. Identify the name of a file to which ENABLE writes the SCREEN COBOL source code by supplying a value for the SCOBOLSOURCE attribute when you generate an application.

2. After making the modifications to the file, compile the source code.

If you want ENABLE to write the SCREEN COBOL source to a file named "srcfile," you can identify this file by entering the following command:

```
   SET APPL SCOBOLSOURCE srcfile !
```

The following list briefly describes some modifications that you can make to the SCREEN COBOL source code.

- Define acceptable values for an input screen field. You can modify the Screen Section of the generated SCREEN COBOL source code by including a MUST BE clause with the appropriate screen field.

- Indicate that lowercase characters are to be translated to uppercase either when they are displayed on the screen or when they are inserted in a data base file. To do this, include the UPSHIFT clause with the appropriate screen field.

- Provide users of the application with the ability to call another application. To do this, you modify the source code to include a call to another requester program. Refer to Section 11 for more information about this type of modification.

- Introduce consistency restraints if you are an experienced SCREEN COBOL programmer. An application generated by ENABLE cannot prevent you from deleting a record from a parent box before you delete the matching record (or records) from the child box. If you delete a parent record before you delete its matching child records, you can no longer read the child records. Therefore, you can no longer use the application to delete these records.

  If this occurs, your data base might contain inconsistent information. To avoid this problem, you could add your own consistency restraints. Adding such restraints is not a trivial task and should not, therefore, be undertaken by anyone who is not an experienced SCREEN COBOL programmer.

- Change the position of fields on the screen, with the following restrictions:

  --Each record must remain contiguous; you cannot intersperse fields from one record with fields from another record.

  --The first and last field of a record must remain in their respective positions; you cannot change the position of these fields within a record.

If you regenerate an application after you have made modifications to the SCREEN COBOL source code, you will have to add those modifications to the regenerated source code.

SECTION 8

USING AN ENABLE APPLICATION

An ENABLE application displays a screen through which you can perform the following tasks:

- Look at the data in one or more data base files
- Update the data in the data base files by making changes to one or more records
- Insert new records in the data base
- Delete records from the data base

To perform these tasks, you press a function key on the terminal keyboard. Figure 8-1 shows the template supplied with ENABLE for T16-651x, T16-652x, and T16-653x terminals. This template indicates the function of each key. Refer to Table 8-1 for a list of the function keys and the operations performed by each.

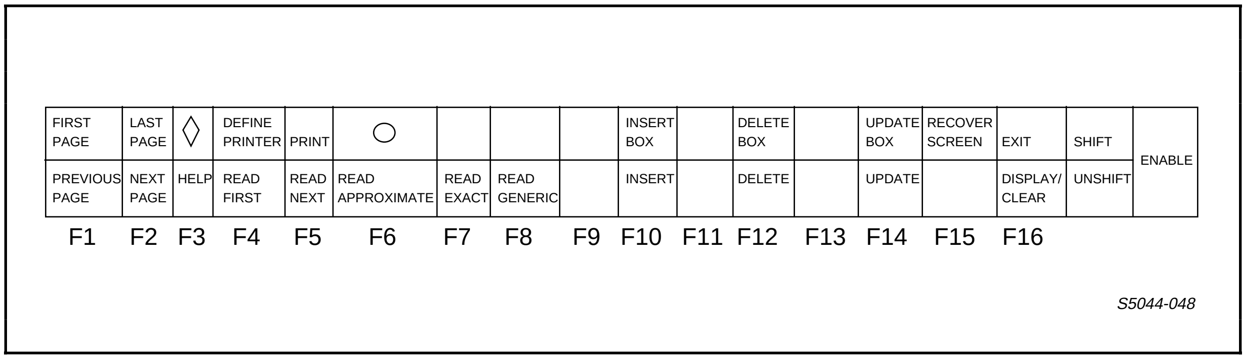
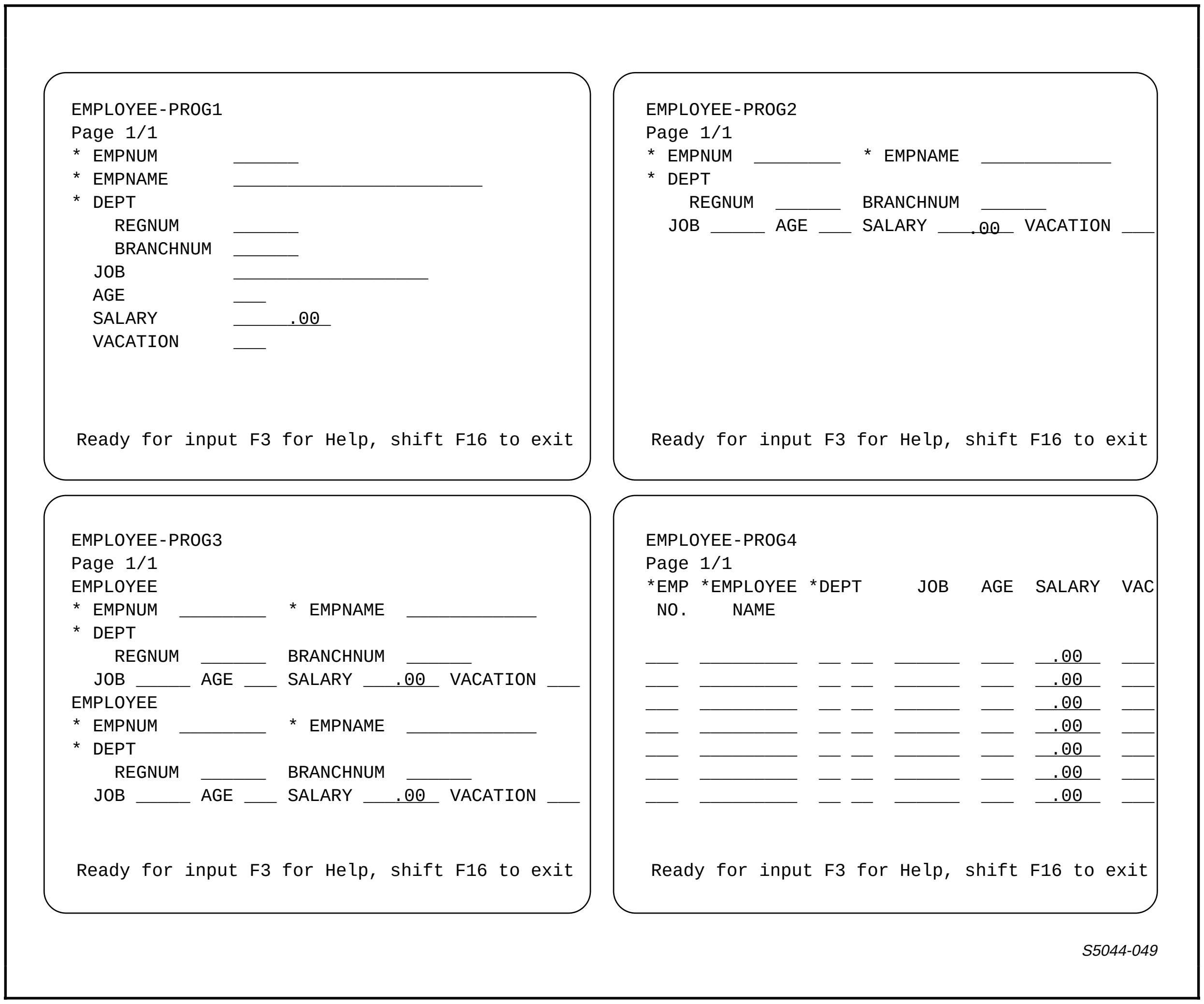


Figure 8-1. ENABLE Template

If the template is available for your type of terminal, it will simplify the process of using an application. Simply place it along the top of the keyboard.

ENABLE DISPLAY SCREENS

An application generated by ENABLE can display a variety of screens. An application that allows you to display or maintain information on employee records might display a screen similar to one of those shown in Figure 8-2.



```
EMPLOYEE-PROG1
Page 1/1
* EMPNUM        ______
* EMPNAME       _______________________
* DEPT
    REGNUM      ______
    BRANCHNUM   ______
  JOB           __________________
  AGE           ___
  SALARY        _____.00_
  VACATION      ___


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

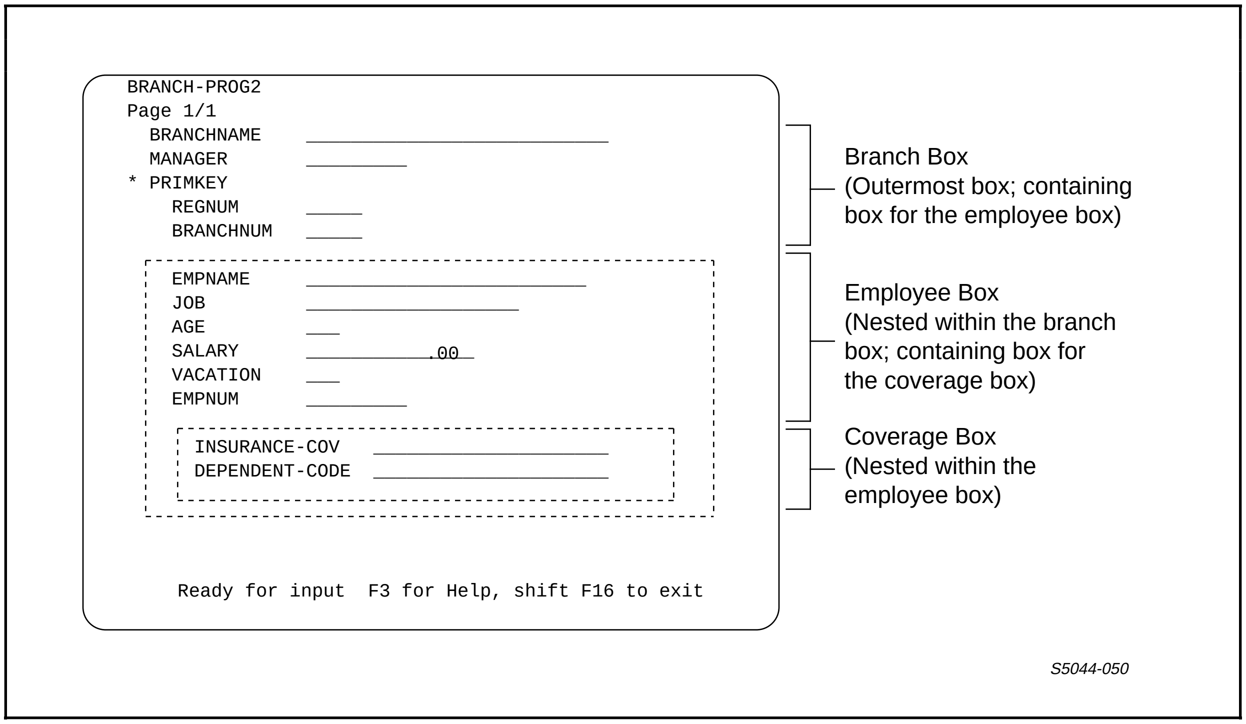
```
EMPLOYEE-PROG2
Page 1/1
* EMPNUM  ________  * EMPNAME  ____________
* DEPT
    REGNUM  ______  BRANCHNUM  ______
  JOB _____ AGE ___ SALARY ___.00__ VACATION ___


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

```
EMPLOYEE-PROG3
Page 1/1
EMPLOYEE
* EMPNUM  ________  * EMPNAME  ____________
* DEPT
    REGNUM  ______  BRANCHNUM  ______
  JOB _____ AGE ___ SALARY ___.00_ VACATION ___
EMPLOYEE
* EMPNUM  ________  * EMPNAME  ____________
* DEPT
    REGNUM  ______  BRANCHNUM  ______
  JOB _____ AGE ___ SALARY ___.00_ VACATION ___


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

```
EMPLOYEE-PROG4
Page 1/1
*EMP *EMPLOYEE *DEPT     JOB   AGE  SALARY  VAC
 NO.    NAME

___  _________  __ __  ______  ___  __.00_  ___
___  _________  __ __  ______  ___  __.00_  ___
___  _________  __ __  ______  ___  __.00_  ___
___  _________  __ __  ______  ___  __.00_  ___
___  _________  __ __  ______  ___  __.00_  ___
___  _________  __ __  ______  ___  __.00_  ___
___  _________  __ __  ______  ___  __.00_  ___


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

Figure 8-2. Sample Display Screens

The edges of the screen form a box around the record (or records) displayed by an application, which is referred to as the outermost box. When an application displays records for more than one file, it shows other boxes nested within this outermost box. Figure 8-3 shows the screen produced by an application that displays records from three files. These records contain information about the divisional branches of a company, the employees who work for each branch, and the type of insurance coverage that each employee has.



```
BRANCH-PROG2
Page 1/1
  BRANCHNAME    ___________________________
  MANAGER       _________
* PRIMKEY
    REGNUM      _____
    BRANCHNUM   _____

    EMPNAME     _________________________
    JOB         ___________________
    AGE         ___
    SALARY      __________.00_
    VACATION    ___
    EMPNUM      _________

      INSURANCE-COV    _____________________
      DEPENDENT-CODE   _____________________

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

Branch Box
(Outermost box; containing box for the employee box)
Employee Box
(Nested within the branch box; containing box for the coverage box)
Coverage Box
(Nested within the employee box)
S5044-050

Figure 8-3. Sample Screen With Nested Boxes

In Figure 8-3, the outermost box displays the fields for a "branch" record. Nested within it is a box that displays the fields for an "employee" record. Because the "employee" box is completely contained within the "branch" box, the "branch" box is called a containing box (as well as the outermost box).

Since the "employee" box is nested within the "branch" box, the "employee" box is called a nested box. The employee box contains another box--the "coverage" box. Because the "employee" box contains the "coverage" box, the "employee" box is a containing box as well as a nested box. The "coverage" box, which does not contain another box, is simply a nested box.

If your application displays several boxes, it is important to recognize the difference between an outermost box, a containing box, and a nested box since:

- You must perform an operation (read or insert a record) on the outermost box before you can perform an operation on any other box.
- You must perform an operation (read or insert a record) on a containing box before you can perform any operation on a box nested within it.

RECORD KEYS

An application uses key fields, or record keys, to identify specific records. You can quickly and efficiently select one record from among thousands stored in a file by indicating a record key. Three categories of record keys exist: primary keys, alternate keys, and courtesy keys. A record cannot have both a primary key and a courtesy key. A record can, however, have a primary key and any number of alternate keys or a courtesy key and any number of alternate keys.

Primary Key

An application can identify a particular record by the contents of a primary key field. If a record has a primary key field of the value 500, for example, you could retrieve that record by requesting the record whose primary key value is equal to 500.

A primary key value is always unique; that is, the primary keys of a file cannot contain duplicate values. Primary keys are always sorted in ascending sequence; this means that if you read through a file sequentially by primary key, the next record is always the record with the next higher primary key value.

Alternate Key

An application can identify a single record or a set of records by the contents of an alternate key field. If, for example, the department number of an employee record is an alternate key, you can retrieve all records of that file sequentially, in department-number order.

Courtesy Key

ENABLE supplies a special courtesy key field for certain types of files. An application can identify a particular record in one of these files by the value of this courtesy key. The courtesy key corresponds to a unique record number associated with each record within a relative, entry-sequenced, or unstructured file.

For a relative file, the record number corresponds to the physical position of the 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 actually been stored in that position.

For an entry-sequenced file, the record number corresponds to the order in which a record is stored in the file. The first record stored has record number 0, the next record stored has record number 1, and so forth. The record numbers are always in ascending order, but the numbering sequence is not always in increments of one. Record 4096, for example, could follow record 32. The last record stored in the file always has the highest record number.

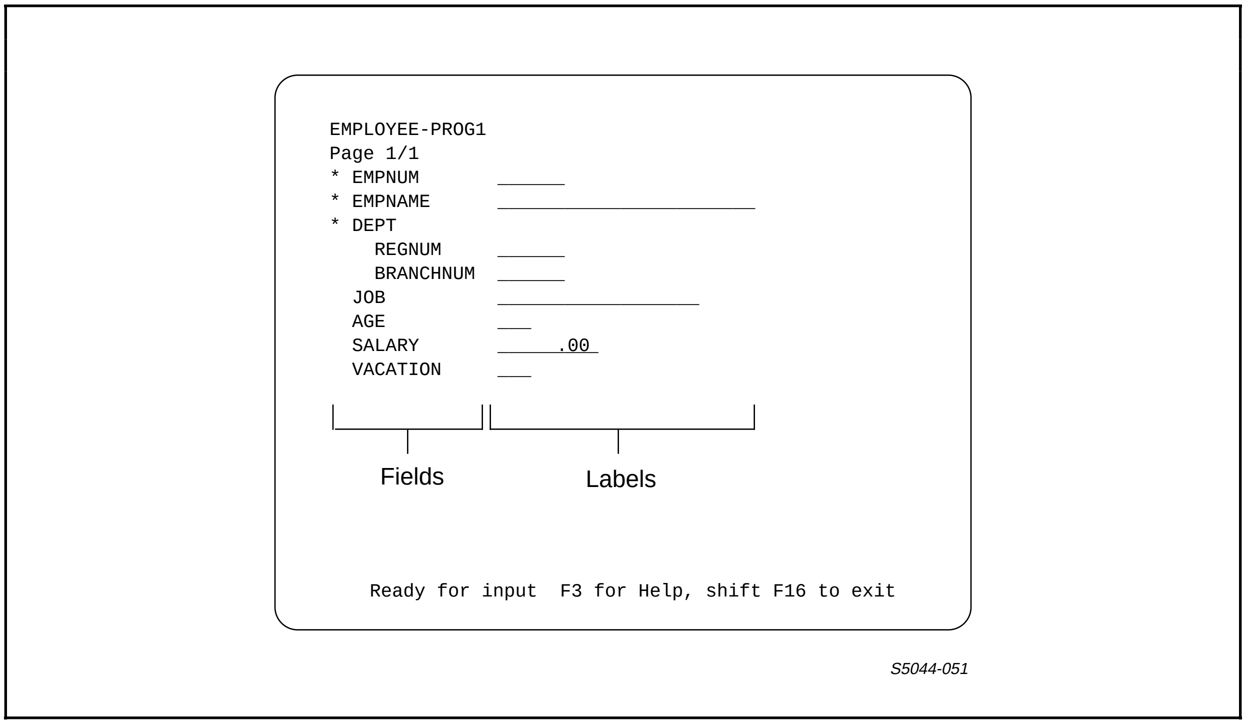
For an unstructured file, the first record is record 0, the next record 1, and so forth. The last record stored in the file always has the highest record number.

When you insert a record with a courtesy key, you do not have to supply a value for that key. Instead, the application can provide a value for you.

As described later under "Labels and Fields," you can visually identify a courtesy key field; however, you cannot visually determine the type of file to which a courtesy key belongs. If you need to determine the file type, ask your data administrator.

LABELS AND FIELDS

The screen produced by an application contains labels and fields. Figure 8-4 shows a sample screen and identifies its labels and fields.



```
EMPLOYEE-PROG1
Page 1/1
* EMPNUM         ______
* EMPNAME        _______________________
* DEPT
    REGNUM       ______
    BRANCHNUM    ______
  JOB            __________________
  AGE            ___
  SALARY         _____.00
  VACATION       ___

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

Figure 8-4. Screen Labels and Fields

Labels

A label appears on the screen to provide information about its associated field. If either an asterisk (*) or a plus sign (+) appears to the left of a label, the label identifies a key field.

If an asterisk appears to the left of a label, the label identifies a primary key field. For example, consider the following label:

```
   * EMPNUM  ____
```

In this example, EMPNUM is the label for a primary key field.

If a plus sign appears to the left of a label, the label identifies an alternate key field. For example, consider the following label:

```
+ EMPNAME __________________
```

In this example, EMPNAME is the label for an alternate key field.

Sometimes a label appears without a field. Such a label identifies a group field. (A group field is a field that can be broken down into smaller items.) When a label identifies a group field, the labels of the items within the group usually appear on the following screen line or lines, indented two screen columns, for example:

```
+ DEPT                       + DEPT
    REGNUM    __      or         REGNUM  __   BRANCHNUM  __
    BRANCHNUM __
```

Notice that in the preceding example, a plus sign appears to the left of the DEPT label, identifying the DEPT group as an alternate key field.

A special label called Record Number identifies a courtesy key field. This label appears as follows:

```
* Record Number ________
```

The asterisk that appears to the left of the label indicates that you can use a Record Number field in the same manner that you use a primary key field.

Fields

An application uses fields to accept and display data from a record. Fields can accept and display the following types of data:

- Numeric data--This type of data consists of digits and possibly a minus sign or decimal point. Consider the following examples of fields that display or accept numeric data:

```
* EMPNUM        ____

  SALARY        ____.00

  VACATION      __

* Record Number ________
```

- Alphanumeric data--This type of data consists of letters of the alphabet, spaces, digits, and special symbols like the hyphen. Consider the following examples of fields that display or accept alphanumeric data:

```
+ EMPNAME     __________________
 BRANCHNAME  _____________
```

- Alphabetic data--This type of data consists of letters of the alphabet and spaces. Consider the following example of a field that displays or accepts alphabetic data:

```
INSURANCE-COV  __________________
```

Notice that, with the exception of fields with decimal points, you cannot visually determine the type of data that a field will accept or display. If the label associated with a field does not clearly define the type of data that you can enter in the field, ask your data administrator for this information. The application will not allow you to enter data of an invalid type in a screen field.

Entering Data in Fields

With the exception of numeric fields that accept signs, you can identify the number of characters (letters or digits) that a field will accept by examining the dashed lines that make up the field. Suppose, for example, that a field appears as follows:

```
+ EMPNAME __________________
```

You can enter characters in this field up to the end of the dashed line. You do not, however, have to enter enough characters to fill the dashed line.

For decimal fields, you must enter the integer portion of the field, the decimal point, and any values that appear to the right of the decimal point. Consider the following field:

```
   SALARY  ____.00
```

If you enter data in this field, it could appear as:

```
   SALARY  34.00__
```

You must include every value that appears to the right of the decimal point when you enter data in a decimal field.

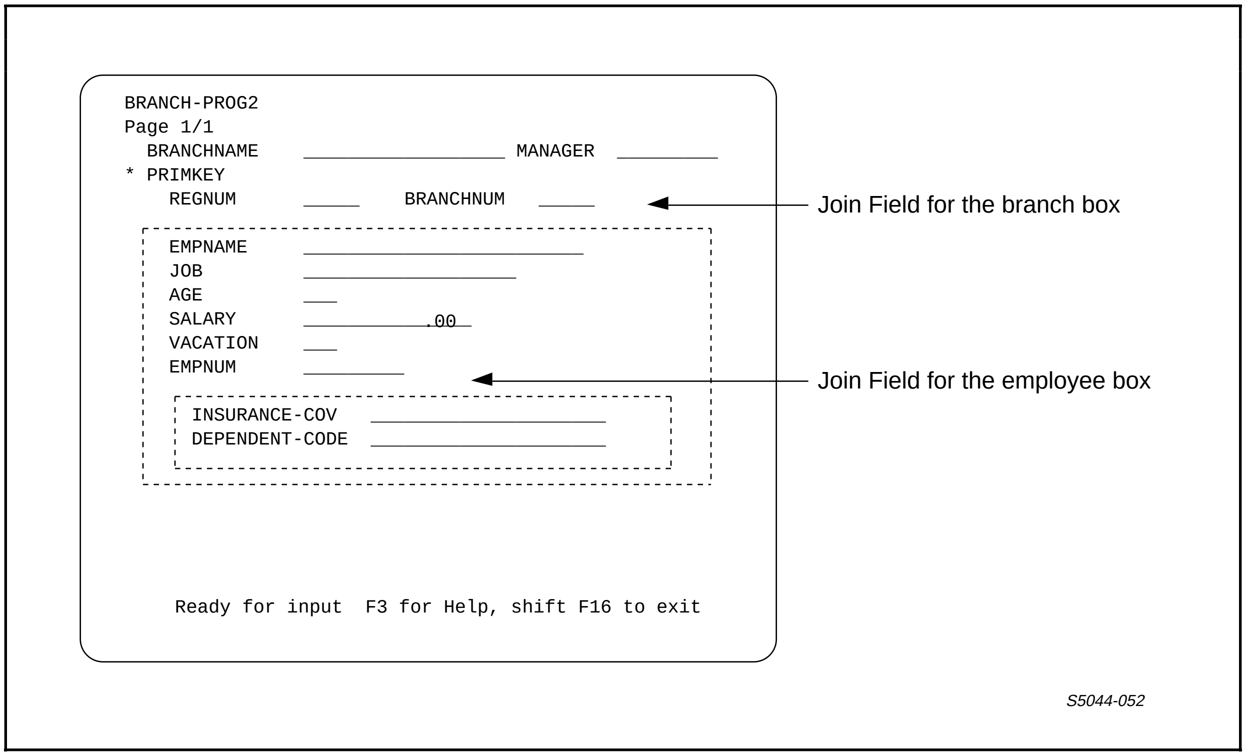
If a numeric field accepts a sign, that field contains an extra character position for the sign. Therefore, you can enter one less digit than the number of dashed lines displayed. Because you cannot visually distinguish a signed field from an unsigned field, ask your data administrator if an application displays signed fields. If the application displays signed fields, you must include a minus sign when you enter a negative value in such fields. (When you enter positive values in these fields do not enter a plus sign; this sign is both unnecessary and illegal).

For alphanumeric or alphabetic fields, you can enter either uppercase or lowercase letters of the alphabet. If you try to read existing records with an operation that involves comparison (READ APPROXIMATE, READ EXACT, or READ GENERIC), follow these rules:

- If the existing records contain uppercase letters, enter uppercase letters when you request a comparison operation.
- If the existing records contain lowercase letters, enter lowercase letters when you request a comparison operation.

Special Fields in Applications That Display Several Boxes

If your application displays several boxes, be aware of special fields whose values connect a containing box to a nested box. These join fields appear directly above the nested box. Figure 8-5 shows a screen with several boxes and identifies the join fields for each containing box.



```
BRANCH-PROG2
Page 1/1
  BRANCHNAME    ____________________ MANAGER  __________
* PRIMKEY
    REGNUM      _____     BRANCHNUM   _____

    EMPNAME     _________________________
    JOB         ___________________
    AGE         ___
    SALARY      __________.00
    VACATION    ___
    EMPNUM      _________

      INSURANCE-COV    _____________________
      DEPENDENT-CODE   _____________________

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

Join Field for the branch box
Join Field for the employee box
S5044-052

Figure 8-5. Join Fields for Containing Boxes

If you cannot identify the join fields for your application, ask your data administrator for this information, since values in these fields can affect the way that you update or delete records.

Cursor Positioning for IBM 327x and T16-651x Terminals

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.

GETTING STARTED

When the ENABLE screen appears on your terminal, you can begin operations. The application initially places the terminal cursor in the first character position of the first field. You can move the cursor to the first character position of the next field by using the TAB key. When you fill a field with characters, the cursor automatically moves to the next field with the exception of key fields. On the T16-652x, T16-653x, and IBM-327x, the cursor does not move to the next field when you completely fill a single key field, the last field of a group key field, or the last field in a nested box.

The following paragraphs describe the operations that you can request to read, insert, delete, or update records. A table at the end of this section provides a brief summary of each of these operations and identifies the function key associated with each operation.

READING RECORDS

You can read a record in a data base file by requesting several different types of read operations. These operations are:

- READ FIRST
- READ NEXT
- READ APPROXIMATE
- READ EXACT
- READ GENERIC

You can request these operations by using function keys F4, F5, F6, F7, and F8.

When you request a read operation, you select a key field with which the application performs the operation. You select this key field by positioning the cursor within the field. If a particular record has only one key, the application assumes that you have selected that key regardless of where you position the cursor.

If you do not position the cursor within a key field, the application assumes that you have selected the first key field that appears on the screen. For this reason, this key field is sometimes called the default key.

If an application displays several boxes, you must position the cursor within the desired box before you request a read operation.

READ FIRST (F4)

When you want to retrieve the first record (or records) in the file according to a selected key field, request a READ FIRST operation. If you select a primary key field, you retrieve the first record actually stored in the file. If you select an alternate key field, you retrieve the first record in the file sorted by the alternate key field. Both primary and alternate keys are sorted in ascending sequence.

If a key field is a group consisting of two fields, a READ FIRST operation retrieves the first record if the group is a primary key field or the record with the lowest group value if the group is an alternate key field. Consider the following group key field:

```
+ DEPT
    REGNUM     __
    BRANCHNUM  __
```

If you request a READ FIRST operation by positioning the cursor in either REGNUM or BRANCHNUM, the application will return the first existing record with the lowest DEPT value.

If duplicate alternate key values exist when you request a READ FIRST operation, the application will return a record with the requested alternate key value and the lowest primary key value.

READ FIRST Operation for an Outermost Box

You can request a READ FIRST operation for an outermost box at any time by pressing F4.

READ FIRST Operation for a Nested Box

To request a READ FIRST operation for a nested box, do the following:

1. Request either a read or insert operation for the containing box.
2. Press F4.

READ NEXT (F5)

If you have established a position within a file by previously requesting another read operation, you can retrieve the next record (or records) in sequence by requesting a READ NEXT operation. Suppose that you request a read by primary key where the key value is 100. Your position within the file is at that record. If you now request a READ NEXT operation, the application will read the next record with a primary key value greater than 100.

If you request an intervening insert, delete, or update operation, you do not lose your position within the file. You can continue the READ NEXT operation as if you had not requested the intervening operation.

After you press a function key to request any read operation, the application displays a message on the upper portion of the screen. This message identifies the key which the application will use if you request a READ NEXT operation.

READ NEXT Operation for an Outermost Box

If you have just performed a READ FIRST, by primary or alternate key, you can read an entire file by requesting repeated READ NEXT operations. To request a READ NEXT operation, press F5.

Refer to the various read functions listed in the table later in this section to determine other uses of the READ NEXT operation.

READ NEXT Operation for a Nested Box

If you have just requested a read operation on a nested box, you can request repeated READ NEXT operations and read any other qualifying records. To request a READ NEXT operation, press F5.

READ APPROXIMATE (F6)

To read the first record (or records) in sequence whose key value is equal to or greater than that of a key value you have entered, request a READ APPROXIMATE operation.

Suppose that you request a READ APPROXIMATE operation for the following group key field:

```
   + DEPT
       REGNUM     __
       BRANCHNUM  __
```

If you enter a value of 01 or 1 in REGNUM, the application will return the first record (or records) with a REGNUM value equal to or greater than 1 (if you enter a leading zero with the 1, the zero is ignored). The BRANCHNUM field of the returned record will be the lowest in value of any records with a REGNUM value of 1.

If you request a READ APPROXIMATE operation on an alphanumeric or alphabetic field, the application will return records in alphanumeric sequence. For example, assume a file contains two records with alternate key name fields: JACK WILSON and JANE ADAMS. If you enter J or JA in the name field and press F6 to request a READ APPROXIMATE operation, the application will return JACK WILSON as the first record.

READ APPROXIMATE Operation for an Outermost Box

To perform a READ APPROXIMATE operation, do the following:

1. Enter a value in the desired key field.

2. Press F6.

READ APPROXIMATE Operation for a Nested Box

For a nested box, you can perform a READ APPROXIMATE operation only if a key field appears in the box. (If a key field does not appear and you request a READ APPROXIMATE operation, the application will perform a READ EXACT operation.) If a key field appears, do the following to request a READ APPROXIMATE operation:

1. Read or insert a record in the containing box.

2. Enter a value in the key field of the nested box.

3. Press F6.

READ EXACT (F7)

To retrieve the first record (or records) with a key field value that exactly matches the one you have entered, request a READ EXACT operation. If you enter a value for a primary key field, the application returns the single existing record. If you enter a value for an alternate key field that has duplicate values, the application retrieves the first record (or records) that has that alternate key value.

If the key field is a group key field, you must enter a value for all the fields in the group. Consider, for example, the following group key field:

```
+ DEPT
    REGNUM     __
    BRANCHNUM  __
```

To request a READ EXACT for the DEPT field, you must first enter a value in both REGNUM and BRANCHNUM. If you enter a 1 in REGNUM and do not enter a value in BRANCHNUM, the application will not find a record unless REGNUM = 1 and BRANCHNUM = 0.

READ EXACT Operation for an Outermost Box

To request a READ EXACT operation, do the following:

1. Enter the entire key value in the desired key field.

2. Press F7.

READ EXACT Operation for a Nested Box

If a key field appears within a nested box, do the following to request a READ EXACT operation:

1. Read or insert a record in the containing box.

2. Enter the entire field value in the desired key field.

3. Press F7.

If a key field does not appear within a box and you request a READ EXACT operation, the application performs the READ EXACT; however, the results will be indistinguishable from a READ FIRST operation.

READ GENERIC (F8)

To retrieve the first record (or records) with a partial key value that matches the one you have entered, request a READ GENERIC operation.

You cannot request a READ GENERIC operation for any key field that contains binary values. The courtesy key field (record number) always contains binary values. Since, with the exception of the courtesy key field, you cannot visually distinguish fields containing binary values from fields containing other values, ask your data administrator to provide this information.

To perform a READ GENERIC, you enter a partial key value and press F8. When you press F8, the application displays the following message at the top of the screen and waits for a response:

```
   Enter compare-length (characters), F8
```

You must then enter the number of characters to be used for the compare operation and press F8 a second time. The application uses the specified number of characters for the compare operation and returns the appropriate record. For example, suppose that you want to request a READ GENERIC operation for the following key field:

```
   + EMPNAME __________________
```

Suppose further that the employee file contains records for employees named MARILYN, MARGARET, and MARIE. If you enter the following partial characters, you will retrieve the indicated records:

M	MARGARET returned	MARIE returned on READ NEXT
MA	MARGARET returned	MARIE returned on READ NEXT
MAR	MARGARET returned	MARIE returned on READ NEXT
MARI	MARIE returned	MARILYN returned on READ NEXT
MARG	MARGARET returned	Record not found on READ NEXT

Now, suppose that a field appears as follows:

```
   * EMPNUM  ____
```

If the employee file contains records for employee numbers 0 through 123, the application will return the record listed in Figure 8-6 depending upon the value you enter and the compare length you specify.

Compare length	Field Value	Record Returned
0	0123	EMPNUM 123 (full field size used)
1	0xxx	EMPNUM 0 (0000)
2	01xx	EMPNUM 100 (0100)
3	012x	EMPNUM 120 (0120)
4	0123	EMPNUM 123 (0123)
no length specified	0123	EMPNUM 123 (full field size used)
value greater than 4	0123	EMPNUM 123 (full field size used)

Figure 8-6. Example of Records Returned

READ GENERIC Operation for an Outermost Box

To request a READ GENERIC operation for an outermost box, do the following:

1. Enter a partial value in the key field.

2. Press F8.

3. Enter the number of compare characters in response to the request from the application.

4. Press F8.

READ GENERIC Operation for a Nested Box

You can request a READ GENERIC operation for a nested box only if a key field appears in that box. If a key field does appear, do the following to request a READ GENERIC operation:

1. Read or insert a record in the containing box.

2. Enter a partial value in the key field.

3. Press F8.

4. Enter the compare length in response to the request from the application.

5. Press F8.

INSERTING RECORDS

You can insert a new record in a file by entering the appropriate values in the record fields and requesting an insert operation. Two insert operations are available:

INSERT--to insert a single record at a time.

INSERT BOX--to insert several records at one time.

When you request an INSERT operation, you must position the cursor within the record to be inserted. When you request an INSERT BOX operation, you must position the cursor within the appropriate box.

INSERT (F10)

To insert a single record into a file, request an INSERT operation. Before you request the INSERT operation, you must enter the appropriate values in the fields of the record to be inserted. If you do not enter values in any of the record fields, the application will issue the following error message:

```
Default record is not acceptable.
```

If all the information necessary to insert a record is not available, you do not have to enter a value in each field of a record. You can update such records later when the information becomes available. You must, however, enter values for the primary key field or any alternate key fields that have been defined as requiring unique values. If you do not enter a value in a nonkey alphanumeric or alphabetic field, the application inserts blanks for these fields. If you do not enter a value in a nonkey numeric field, the application inserts zeros for this field.

If you request an INSERT operation for an entry-sequenced or unstructured file, the application ignores any value that you enter in a Record Number field. The application and the computer system automatically supply the appropriate number for the record.

If you request an INSERT operation for a relative file, the application uses the value that you enter in the Record Number field. The application handles values in this field as follows:

No value (or no change to the previously displayed value)	The application inserts the record in the first available position in the file.
Negative value	The application inserts the record at the end of the file.
Positive value	The application inserts the record at that numbered position in the file if possible; if that is not possible, the application issues a Duplicate Key error message.

INSERT Operation for an Outermost Box

To insert a single record for an outermost box, do the following:

1. Enter appropriate values in the fields of the record.

2. Press F10.

Figure 8-7 illustrates a sample record INSERT operation for an application that displays one box. The file into which this record is inserted is key-sequenced, as determined by the existence of a primary key (* EMPNUM) and alternate keys (+ EMPNAME and + DEPT).

```
-----------------------------------------------------------------
|                                                               |
|  STEP 1:  Enter the values in the fields.                     |
|                                                               |
|                             Page 1/1                          |
|    (Stored as 0387)      * EMPNUM       387_                  |
|                          + EMPNAME      JANE WILSON_________   |
|    (Stored as 0600)      + DEPT                               |
|      (1st two digits 06)     REGNUM     6_                     |
|      (2nd two digits 00)     BRANCHNUM  __                     |
|    (Stored as ANALYST)      JOB          __ANALYST_           |
|                             AGE          22                    |
|                             SALARY       1600.00               |
|    (If this is a signed     VACATION     1_                    |
|    field, the value is                                        |
|    stored as 1--the                                           |
|    first position on                                          |
|    the display is                                             |
|    reserved for the                                           |
|    sign; if this is                                           |
|    an unsigned field,                                         |
|    the value is stored                                        |
|    as 01.)                                                    |
|                                                               |
|  STEP 2:  Press F10 to insert the record.                     |
|                                                               |
-----------------------------------------------------------------
```

Figure 8-7. Inserting a Record

Figure 8-7 illustrates the following:

- You do not have to enter a value in each field.

- You can enter characters in any position on the line. The application will store values entered in numeric fields like EMPNUM right-justified with leading zeros. The application will store values entered in alphanumeric fields like JOB left-justified with trailing blanks.

INSERT Operation for a Nested Box

To insert a single record for a nested box, you must do the following:

1. Request a read or insert operation for the containing box.

2. Enter the appropriate values in the fields of the nested box.

3. Position the cursor within the record to be inserted if more than one record exists in the box.

4. Press F10.

Figure 8-8 illustrates a sample record INSERT operation for an application that displays several boxes.

```
----------------------------------------------------------------
|                                                              |
|  STEP 1:  Enter values in the fields of the containing box.  |
|                                                              |
|    BRANCH-PROG1                                              |
|    Page 1/1                                                  |
|      BRANCHNAME  San Francisco_                              |
|      MANAGER  23__                                           |
|    * PRIMKEY                                                 |
|        REGNUM  _1                                            |
|        BRANCHNUM  _2                                         |
|    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ |
|    | EMPNUM  ____  EMPNAME ______________  JOB ___________ | |
|    | AGE     __   SALARY ____.00    VACATION __           | |
|    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ |
|                                                              |
|  STEP 2:  Press F10 to insert the record in the containing   |
|           box.                                               |
|                                                              |
|  STEP 3:  Enter values in the fields of the nested box.      |
|                                                              |
|    BRANCH-PROG1                                              |
|    Page 1/1                                                  |
|      BRANCHNAME  San Francisco_                              |
|      MANAGER  23__                                           |
|    * PRIMKEY                                                 |
|        REGNUM  _1                                            |
|        BRANCHNUM  _2                                         |
|    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ |
|    | EMPNUM  0123  EMPNAME Joe Jones_____  JOB Salesman___ | |
|    | AGE    39   SALARY 3000.00    VACATION 1_            | |
|    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ |
|                                                              |
|  STEP 4:  Press F10 to insert the record in the nested box.  |
|                                                              |
----------------------------------------------------------------
```

Figure 8-8. Inserting a Record in a Nested Box

NOTE

Do not enter values in a nested box until you request a read or insert operation for its containing box. If you enter values in the nested box before you request an operation for its containing box, the application will remove the values from the nested box when it performs the operation on the containing box. If an application automatically reads a record for the nested box whenever you request an operation for the containing box, the application will replace the values in the nested box with values read from the data base.

INSERT BOX (Shifted F10)

If you want to insert several records at one time, request an INSERT BOX operation. Before you request the INSERT BOX operation, you must enter values in some fields of at least one record. When the application successfully completes the INSERT BOX operation, it displays a message indicating the number of records inserted.

If the application encounters an error during an INSERT BOX operation, it highlights the fields of the record in error. The application displays the following prompt on line 2 of the screen:

```
Continue processing, Y/N, F1
```

If the application displays this prompt, you must enter either the character Y (to indicate that you want to continue) or the character N (to indicate that you want to stop) and then press F1. If you enter a Y, the application continues the INSERT BOX operation beginning with the record directly following the record in error. If you enter an N, the application stops the INSERT BOX operation. In this case, the application has already inserted any records that precede the record in error.

You can reverse the effects of an INSERT BOX operation by requesting an UNDO (SF13) operation immediately after the INSERT BOX operation.

INSERT BOX Operation for an Outermost Box

You can insert several records in an outermost box by doing the following:

1. Entering the appropriate values in the fields of the records.

2. Pressing shifted F10 to insert the records.

Figure 8-9 illustrates a sample INSERT BOX operation for an application that displays a single box. You do not have to enter values for every existing record position.

```
-----------------------------------------------------------------
|                                                               |
|  STEP 1:  Enter values in the record fields.                  |
|                                                               |
|  Page 1/1                                                     |
|  * EMP    + EMPLOYEE    + DEPT       JOB     AGE   SALARY VAC |
|    NO.       NAME                                             |
|   323_  Joe Smith______  2_  3_  Manager____  37  4000.00  12 |
|   239_  James King_____  2_  3_  Salesman___  39  2500.00  9_ |
|   245_  Sally Jones____  2_  3_  Programmer_  29  3900.00  13 |
|   ____  _______________  __  __  ___________  __  ____.00  __ |
|   ____  _______________  __  __  ___________  __  ____.00  __ |
|   ____  _______________  __  __  ___________  __  ____.00  __ |
|   ____  _______________  __  __  ___________  __  ____.00  __ |
|                                                               |
|  STEP 2:  Press shift F10 to insert the records.              |
|                                                               |
|  The application displays the following message after        |
|  completing the INSERT BOX operation:                         |
|                                                               |
|     Items inserted:  003                                      |
|                                                               |
-----------------------------------------------------------------
```

Figure 8-9. Example of INSERT BOX Operation for a Single Box

INSERT BOX Operation for a Nested Box

To request an INSERT BOX operation for a nested box, do the following:

1. Read or insert a record for the containing box.

2. Enter screen values in the nested box.

3. Press shifted F10.

NOTE

Do not enter values in a nested box unless you have previously requested a read or insert operation for the containing box. If you enter values in the nested box before you request an operation for the containing box, the application will remove the values from the nested box when it performs the operation on the containing box. If an application automatically reads a record for a nested box whenever you request an operation for a containing box, the application will replace the values in the nested box with values read from the data base.

Figure 8-10 illustrates an INSERT BOX operation for an application that displays several boxes.

```
-----------------------------------------------------------------
|                                                               |
|  STEP 1:  Enter values in the fields of the containing box.   |
|                                                               |
|  BRANCHNAME  San Jose______   MANAGER ____                    |
|  * PRIMKEY                                                    |
|    REGNUM     __    BRANCHNUM __                              |
|  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ ~~~~~~~~~~~~~~~~~~~~   |
|  | EMP       EMPLOYEE          JOB       AGE  SALARY  VAC |   |
|  | NO.         NAME        DESCRIPTION                    |   |
|  | ____  __________________  ____________  __  ____.00  __ |   |
|  | ____  __________________  ____________  __  ____.00  __ |   |
|  | ____  __________________  ____________  __  ____.00  __ |   |
|  | ____  __________________  ____________  __  ____.00  __ |   |
|  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   |
|                                                               |
|  STEP 2:  Press F6 to read the record.                        |
|                                                               |
|  STEP 3:  Enter values in the fields of the nested box.       |
|                                                               |
|  BRANCHNAME  San Jose______   MANAGER 359_                    |
|  * PRIMKEY                                                    |
|    REGNUM     1_    BRANCHNUM 3_                              |
|  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   |
|  | EMP       EMPLOYEE          JOB       AGE  SALARY  VAC |   |
|  | NO.         NAME        DESCRIPTION                    |   |
|  | 359_  Johnny Joe Myers__  Manager_____  53  4500.00  23 |   |
|  | 595_  Betty Lee_________  Secretary___  27  1900.00  10 |   |
|  | 323_  Sylvia Smitty_____  Saleswoman__  31  4200.00  20 |   |
|  | 534_  Philip Peters_____  Clerk_______  32  1500.00  _3 |   |
|  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   |
|                                                               |
|  STEP 4:  Press shifted F10 to insert the records in the      |
|           nested box.                                         |
|                                                               |
|  When the application sucessfully completes the INSERT BOX    |
|  operation, it displays:                                      |
|                                                               |
|     Items inserted:  004                                      |
|                                                               |
-----------------------------------------------------------------
```

Figure 8-10. INSERT BOX Operation for a Nested Box

DELETING RECORDS

You can delete only those records that are currently displayed on the screen. To retrieve records that you want to delete, use any suitable read operation. After you retrieve a record (or records), you can request the following delete operations:

DELETE--to delete a single record

DELETE BOX--to delete several records

When you request a DELETE operation, you must position the cursor within the record to be deleted. When you request a DELETE BOX operation, you must position the cursor within the appropriate box.

If you request a delete operation and then change your mind, you can use the UNDO key (SF13) to replace the deleted record.

DELETE (F12)

If you want to delete a single record, request a DELETE operation. To request a DELETE operation, do the following:

1. Perform any read operation to retrieve the record you want to delete.

2. Position the cursor over the record you want to delete if more than one record appears in the box.

3. Press F12 to delete the record.

DELETE BOX (Shifted F12)

If you want to delete several records at the same time, request a DELETE BOX operation. You can only delete those records that appear on the screen. To request a DELETE BOX operation, do the following:

1. Retrieve the records that you want to delete by requesting any suitable read operation.

2. If any records appear that you do not want to delete, replace these records with the default record.

3. Press Shifted F12 to delete the records shown on the screen.

If the application encounters an error during the DELETE BOX operation, it highlights the fields of the record in error. The application then displays the following prompt at the top of the screen:

```
   Continue processing, Y/N, F1
```

If the application displays this prompt, you must enter either the character Y (to indicate that you want to continue) or the character N (to indicate that you want to stop) and then press F1. If you enter a Y, the application continues the DELETE BOX operation beginning with the record that directly follows the record in error. If you enter an N, the application stops the DELETE BOX operation. In this case, the application has already deleted any records that appear above the record in error. If you request an UNDO operation at this point, however, you can return the data base to its previous condition.

DELETE Considerations for Applications With Several Boxes

If you delete a record from a containing box, you can no longer use the application to retrieve the matching record (or records) for the nested box. If this is a problem for your application, do the following:

1. Use any read operation to retrieve the record for the containing box.

2. Use an appropriate read operation to retrieve the record (or records) for the nested box.

3. Delete the record (or records) displayed in the nested box.

4. Repeat steps 2 and 3 until the application does not return any records as a result of the read operation.

5. Delete the record in the containing box.

UPDATING RECORDS

You can only update a record that is currently displayed on the screen. To retrieve the record (or records) you want to update, use any suitable read operation. After you have retrieved the record (or records), you can request the following update operations:

UPDATE--to update a single record at a time

UPDATE BOX--to update several records at a time

If you request an update operation, you must position the cursor within the record being updated. If you request an UPDATE BOX operation, you must position the cursor with the appropriate box.

When you request an update operation, you can change the primary key of a record in a key-sequenced file. If that record has a unique alternate key value, however, you must also change that alternate key.

CAUTION

If you retrieve a numeric field value and the data value stored in that field is invalid, the application issues an error message under certain circumstances and displays zeros in the field. If you immediately press F14, the application stores the zeros in the record. Since data administrators often wish to examine any invalid data, you may want to avoid requesting an update operation should the application indicate that invalid numeric is encountered. In this event, the application highlights the fields containing invalid data.

UPDATE (F14)

To change a single record, request an update operation by:

1. Use any read operation to retrieve the record.
2. Make the necessary changes to the fields of the record.
3. Position the cursor within the record to be updated if more than one record appears in the box.
4. Press F14 to update the record.

UPDATE BOX (Shifted F14)

When you want to change several records at one time, request an UPDATE BOX operation. (You can only update those records that appear on the screen.) To request an UPDATE BOX operation, do the following:

1. Use any read operation to retrieve the records.
2. Make the necessary changes to the fields of the records.
3. Press shifted F14 to update the records.

If the application encounters an error during the UPDATE BOX operation, it highlights the fields of the record in error. The application then displays the following prompt at the top of the screen:

```
Continue processing, Y/N, F1
```

If the application displays this prompt, you must enter either the character Y (to indicate that you want to continue) or the character N (to indicate that you want to stop) and then press F1. If you enter a Y, the application continues the UPDATE BOX operation beginning with the record that directly follows the record in error. If you enter an N, the application stops the UPDATE BOX operation. In this case, the application has already updated any records that appear above the record in error. You can, however, request an UNDO operation at this point and return the data base to its previous condition.

UPDATE Considerations for Applications With Several Boxes

If you request an update operation that changes the value of the join field of a containing box, you can no longer use the application to read any records in the nested box that have that same join-field value. If this is a problem for your application, do the following:

1. Request a read operation to retrieve the containing record.
2. Request a read operation to retrieve the nested record (or records).
3. Delete the nested record (or records).
4. Repeat steps 2 and 3 until the application does not return a record as a result of the read operation.
5. Change the necessary fields of the containing record.
6. Press F14 to update the record in the containing box.
7. Replace the nested record (or records) by requesting the appropriate insert operations for the nested box.

If many nested records need to be deleted and replaced, consider asking your data administrator for two single-file applications that you can use to change the join field-values of these records.

UNDOING AN INSERT, DELETE, OR UPDATE (SHIFTED F13)

You can reverse the effects of an INSERT, DELETE, or UPDATE operation by requesting an UNDO operation. You request an UNDO operation by pressing shifted F13.

Undoing a DELETE Operation

To undo a DELETE operation, the application must insert the deleted record (or records). The application can insert the deleted record if you have not requested another delete, insert, update, or read operation since you requested the delete operation you want to reverse and another application has not inserted the deleted record.

You can undo a DELETE operation even if your application does not support insert operations.

Undoing an UPDATE Operation

To undo an UPDATE operation, the application must replace the old record (or records). The application can replace the old record if the following is true:

- You have not requested another DELETE, INSERT, READ, or UPDATE operation since you requested the update operation you want to reverse.

- Another application has not modified the updated record.

You can undo an UPDATE operation even if you changed the primary key.

Undoing an INSERT Operation

To undo an INSERT operation, the application must delete the record (or records) you inserted. The application can reverse the insert operation if the following is true:

- You have not requested another DELETE, INSERT, READ, or UPDATE operation since you requested the INSERT operation you want to reverse.

- Another application has not changed or deleted the inserted record.

- You did not insert the record (or records) in an entry-sequenced or unstructured file. The computer system will not allow the application to delete a record from either an entry-sequenced or an unstructured file.

You can undo an INSERT operation even if your application does not support delete operations.

PERFORMING SPECIAL SCREEN OPERATIONS

Several types of screen operations are available. The operations are:

- Defining a printer
- Printing a screen image
- Recovering a display screen

Defining a Printer (Shifted F4)

To define a printer, press shifted function key F4. The application issues a prompt that requests you to specify where printing should be directed. Type in the name of the appropriate device. The device can be a terminal, printer, or process such as the Tandem spooler. The device cannot be a disc file.

Printing a Screen Image (Shifted F5)

To send an image of the current screen to a device, press shifted function key F5. The application sends the screen image to the device you previously specified in response to the shifted F4 key prompt.

Recovering a Display Screen (Shifted F15)

To recover a screen image, press shifted function key F15. This function can be used when unexpected problems occur, for example, when someone inadvertently turns off your terminal. Note that the contents of the screen are not necessarily the same as they were before recovery was initiated.

OPERATOR DISPLAY AND ERROR MESSAGES

The application displays messages on the upper and lower portions of the screen. For example, when the application is ready for you to perform an operation, the following message is displayed on the lower portion of the screen:

```
Ready for Input
```

After you perform an initial read operation for a box, one of the following messages appears on the upper portion of the screen whenever you are positioned within the box:

Approx Key key-name	appears after a READ FIRST, READ NEXT, and READ APPROXIMATE.
Exact Key key-name	appears after a READ EXACT.
Generic Key key-name	appears after a READ GENERIC.

Each time you successfully complete an operation, an appropriate message is displayed on the lower portion of the screen:

- Record deleted OK
- Delete undone OK
- Record inserted OK
- Update undone OK
- Record read OK
- Items inserted: nnn
- Record updated OK
- Items deleted: nnn
- Insert undone OK
- Items updated: nnn

The application displays error messages on the last line of the screen; the messages persist until you press a function key. These messages are listed in Appendix B under "Application Run-Time Messages."

TERMINAL FUNCTION KEYS

To perform operations with an application, you use the terminal function keys located across the top of the keyboard. Table 8-1 lists these function keys and their specific operations. Entries in the function key column indentify the function keys for the T16-651x, T16-652x, and T16-653x terminals. The comparable program function and attention keys for IBM-327x terminals appear in parentheses.

If you are using an IBM-327x terminal, the ENTER key serves as a shift key for twelve operations corresponding to the Tandem shifted F1, F2, F3, F4, F5, F6, F10, F12, F13, F14, and F15 keys. To use these corresponding shifted functions, do the following:

1. Press the ENTER key first. The screen displays the word SHIFT, and the PF keys take on their secondary functions. To cancel a SHIFT operation before execution, press the ENTER key again.

2. Press the appropriate PF key. It is not necessary to hold the ENTER key down when the PF key is pressed.

Table 8-1. Terminal Function Keys (Continued next page)

Function Key	Template Label	Operation
F1 (PF1)	PREVIOUS PAGE	Display the previous screen page or HELP screen.
F1 shifted (ENTER,PF1)	FIRST PAGE	Display the first screen 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 image screen and HELP screens. HELP screens display all function key assignments.
F3 shifted (ENTER, PF3)	/\ \/	Application-dependent; often calls another application. This function key might not be supported by your application.
F4 (PF4)	READ FIRST	Retrieve the first record or records in a file sorted by a particular key field. You can then use F5 to continue reading.
F4 shifted (ENTER, PF4)	DEFINE PRINTER	Indicate where printing of the display screen is to be directed. You can then use shifted F5 to print the screen.

Table 8-1. Terminal Function Keys (Continued)

Function Key	Template Label	Operation
F5 (PF5)	READ NEXT	Retrieve the next record or records in sequence according to the operation in progress.
F5 shifted (ENTER,PF5)	PRINT	Send an image of the current screen to the device specified by shifted F4.
F6 (PF6)	READ APPROX	Retrieve the first record or records whose key value is greater than or equal to that of the value you have entered. You can then use F5 to continue reading.
F6 shifted (ENTER, PF6)		Application dependent; function might not be supported.
F7 (PF7)	READ EXACT	Retrieve the record or records whose key value matches that of the value you have entered. You can then use F5 to continue reading as long as more records with the same key value exist.
F8	READ GENERIC	Retrieve the first record or records whose partial key value matches that of the partial key value you have entered. You can then use F5 to continue reading sequentially. You cannot use F8 on the courtesy key (record number) field.

Table 8-1. Terminal Function Keys (Continued)

Function Key	Template Label	Operation
F10 (PF10)	INSERT	Insert the record under the cursor.
F10 shifted (ENTER, PF10)	INSERT BOX	Insert all records currently displayed in the box which contains the cursor.
F12 (PF11)	DELETE	Delete the record currently displayed on the screen under the cursor. You cannot use F12 with records from entry-sequenced or unstructured files.
F12 shifted (ENTER, PF11)	DELETE BOX	Delete all the records currently displayed in the box with the cursor. You cannot use F12 with records from entry-sequenced or unstructured files.
F13 shifted (ENTER, PF7)	UNDO	Reverse the immediately preceding insert, delete, or update operation.
F14 (PF12)	UPDATE	Change the values of the record currently displayed under the cursor.
F14 shifted (ENTER, PF12)	UPDATE BOX	Change the values of all the records currently displayed in the box with the cursor.

Table 8-1. Terminal Function Keys (Continued)

Function Key	Template Label	Operation
F15 (ENTER, PF 9)	RECOVER	Redisplay data items on the screen. Items displayed on the screen are not necessarily those that were on the screen at the time the key was pressed.
F16	DISPLAY/CLEAR	Display or clear the current record--the record most recently read from or written to the data base.
F16 shifted (PA2)	EXIT	Exit the program. The message "APPLICATION TERMINATING" appears on the lower portion of the screen.
NEXT PAGE		Display the next screen page or HELP screen.
PREV PAGE		Display the previous screen page or HELP screen.

SECTION 9

RESOLVING PROBLEMS

This section provides guidelines to help you when you encounter problems when generating or using an application. The problems discussed in this section include:

- Handling extended memory overflow
- Difficulty accessing a dictionary or a record description
- Applications that fail to run or that fail to run properly

HANDLING PROBLEMS WITH EXTENDED MEMORY OVERFLOW

ENABLE uses an area of extended memory for its internal tables. When you start ENABLE, the GUARDIAN operating system allocates 500 pages for these tables. The amount of available space decreases as you describe and add boxes and applications. When this amount becomes dangerously low, ENABLE issues an warning message stating:

```
   Very low on extended memory, delete unnecessary objects
```

If you receive this message when you are using ENABLE interactively, do the following to reclaim space in these tables:

1. Use the INFO APPL *, SUMMARY command to obtain a list of the boxes and applications that currently exist in the object buffer.

2. Use the DELETE APPL command to delete all applications that you have already generated. Use the DELETE BOX command to delete all of the boxes associated with these applications. You must delete an application before you can delete a box used by the application.

3. If you have not generated any applications, you must delete one or more applications (and the associated boxes) before you can generate the remaining applications.

4. If several applications use the same boxes, deleting one or more applications might not reclaim sufficient space. In this case, you should:

   --Use the OUT, INFO, and SHOW commands to obtain a listing of the current contents of the object buffer.

   --Exit from ENABLE.

   --Increase the number of extended memory pages allocated by entering the command Interpreter PARAM command and specifying a value for the EXTPAGES parameter. (Refer to the _ENABLE Reference Manual_ for more information about this parameter.)

   --Restart ENABLE and generate your applications.

If you are using ENABLE noninteractively by entering commands in a command file, you will not encounter this error until ENABLE terminates with the following error message:

```
ENABLE tables overflow allocated extended memory
```

In this case, you should:

1. Increase the number of extended memory pages allocated by entering the command Interpreter PARAM command to specify a value for the EXTPAGES parameter.

2. Resubmit your command file to ENABLE.

RESOLVING PROBLEMS ACCESSING A DICTIONARY

ENABLE accesses a dictionary to obtain the record description for a particular data base file. If you are using ENABLE interactively and receive an error message stating that the dictionary either cannot be found or cannot be opened, do the following:

1. Use the SHOW command to determine the current value of the DICTIONARY attribute.

2. Make sure that the value of the DICTIONARY attribute is set to the system, volume, and subvolume where the dictionary resides.

3. If the current value of the DICTIONARY attribute is correct, exit from ENABLE and use the FUP INFO command to check the security attributes of the dictionary. Check the following:

   --If you do not own the dictionary, make sure that the security attributes allow you to access the dictionary. If you do not have access, ask the owner of the dictionary to change its security attributes.

   --If you are using a dictionary that resides on another system, make sure that the dictionary is secured for network access and that you have appropriate network passwords on both systems.

If the application was generated with an obey file, make sure that the obey file refers to the proper system, volume and subvolume for the dictionary associated with each box. This may mean including a SET DICTIONARY command for each box to be described.

RESOLVING PROBLEMS THAT OCCUR DURING APPLICATION EXECUTION

If an ENABLE application fails to run, or if a running application fails to perform properly:

- Check the PATHMON name you have selected. Make sure the name begins with a dollar sign, has no more than five characters, and is unique within your system.

- Make sure the PATHMON name you have selected is not the name of a running process. If it is the name of a running process, stop the process and change the PATHMON name or see your data administrator.

- Make sure the PATHCOM command file you are using is correct.

If the application still fails to run after you have checked each of the above, regenerate the application and PATHCOM command file. Run PATHMON using the generated PATHCOM command file. If the program still fails to run, see your data administrator.

SECTION 10

MAINTAINING AN APPLICATION

This section provides guidelines to help you:

- Move an application from one place to another
- Reclaim disc space used by old versions of applications

MOVING AN APPLICATION

For various reasons, you might want to move an application from one system, volume, or subvolume to another system, volume, or subvolume. The following paragraphs contain two sets of guidelines for moving an application. Follow the first set of guidelines if you want to move an existing application. Follow the second set of guidelines if you are generating an application that is to be moved.

Moving a Generated Application

To move a generated application from one system, volume, or subvolume to another, do the following:

1. Use the SCREEN COBOL Utility Program (SCUP) to move either the object file containing the SCREEN COBOL object code or the object code of a particular application. Refer to the PATHAID Reference Manual for more information about moving SCREEN COBOL object files.

2. Use either the text editor or the FILE UTILITY PROGRAM (FUP) to move the PATHCOM command file.

3. After you have moved the PATHCOM command file, check the system, volume, or subvolume references in the following commands:

   - SET TCP TCLPROG

   - SET SERVER ASSIGN

   If these commands refer to the old system, volume, or subvolume, change these references.

4. If necessary, use FUP to move the data base file (or files), and its associated alternate key file. Use the FUP ALTER command to adjust the reference to the location of the alternate key file. Refer to the GUARDIAN Operating System User's Guide for information about the FUP ALTER command.

Generating an Application To Be Moved

If you are generating an application that you intend to move to another system, volume, or subvolume, do the following:

1. Obtain the SCREEN COBOL source code by setting a value for the SCOBOLSOURCE attribute before you add the application.

2. Suppress compilation of the SCREEN COBOL object code by setting the SCOBOLOBJECT attribute to a null value. For example:

   ```
   SET APPL SCOBOLOBJECT
   ```

3. Use FUP or the editor to move the SCREEN COBOL source code to the desired system, volume, or subvolume.

4. Compile the SCREEN COBOL source code by using the following command:

```
SCOBOLX/ IN <source-file>, OUT <list-file>, MEM 64 /

  <object-ID>
```

<source-file>

is the name of the edit-type file containing the SCREEN COBOL source code.

<list-file>

is the name of the file to which the SCREEN COBOL listing is to be directed.

<object-ID>

is the name of the SCREEN COBOL object file identifier, typically POBJ, which can be up to five characters long.

5. Use FUP or the editor to move the PATHCOM file.

6. Edit the PATHCOM command file and replace the question marks (???) that appear after the SET TCP TCLPROG command with the name of the SCREEN COBOL object file. If you are going to move a data base file, be sure that the SET SERVER ASSIGN commands identify the proper location of that file.

7. If necessary, use FUP to move the data base file (or files), including any alternate key files associated with the data file. After moving the alternate key files, use the FUP ALTER command on the data file to adjust the reference to the location of the alternate key file. Refer to the GUARDIAN Operating System User's Guide for information about the FUP ALTER command.

RECLAIMING DISC SPACE

When you generate an application, ENABLE calls the SCREEN COBOL compiler to compile the SCREEN COBOL source code. If you generate several applications with the same name and direct these applications to the same object files, the SCREEN COBOL compiler adds the new version of the object code to the object files but does not purge the old version of the object code. Because the old version takes up disc space, you might want to remove unnecessary versions.

To reclaim the disc space used by an old version of an application, use the SCREEN COBOL Utility Program (SCUP). When you use SCUP, you must know the names of the code file and directory file to which the SCREEN COBOL compiler writes the object code. If you set the SCOBOLOBJECT attribute to a value when you generate an application, the SCREEN COBOL compiler uses this value to determine the name of the code file and the directory file. If you do not set a value for the SCOBOLOBJECT attribute, the SCREEN COBOL compiler uses the starting value of this attribute (POBJ) to determine the names of these files. In either case, the object file names take the following forms:

- <SCOBOLOBJECT-value>COD

  for the code file name; for example, "pobjcod"

- <SCOBOLOBJECT-value>DIR

  for the directory file name; for example, "pobjdir"

You can use SCUP to reclaim disc space by performing the tasks described in Figure 10-1. This figure also shows an example of each task and describes the events that occur when you perform each task.

STEP 1: Call SCUP by entering the SCUP command from the command interpreter.

```
:SCUP
```

SCUP displays the product identification message and issues a prompt.

```
SCREEN COBOL UTILITY - T9103E01 - (01MAR82)
?
```

STEP 2: Enter the SCUP VOLUME command to set the default volume and subvolume.

The following example identifies "$data" as the default volume and "enab" as the default subvolume:

```
?VOLUME $data.enab
```

STEP 3: Enter the FILE command to set the default object file name.

The following example identifies "pobj" as the default object file name:

```
?FILE pobj
```

STEP 4: Enter the INFO command to display all versions of all programs in the object file as follows:

```
?INFO (*(*))
```

Figure 10-1. Using SCUP to Reclaim Disc Space
(Continued next page)

When this command is entered, SCUP displays:

```
$DATA.ENAB.POBJ
PROGRAM (VERSION)     ACCESS SIZE  COMPILATION VER/DATE
EMPLOYEE-PROG (1)    ON    4096  (E06) 04 SEPT 1983  11:43:14
EMPLOYEE-PROG (2)    ON    4096  (E06) 29 SEPT 1983  20:17:16
EMPLOYEE-PROG (3)    ON    4096  (E06) 10 JAN 1984   13:40:18
EMPLOYEE-PROG (4)    ON    4096  (E06) 15 FEB 1984   15:23:32
```

STEP 5: Enter the DELETE command to delete all versions except for the latest version from the directory file.

In this example, the DELETE command deletes all versions except for the latest version of "employee-prog."

```
   ?DELETE (employee-prog (+))
```

SCUP displays:

```
   $DATA.ENAB.POBJ
   DELETED EMPLOYEE-PROG (1)
   DELETED EMPLOYEE-PROG (2)
   DELETED EMPLOYEE-PROG (3)
```

To delete all but the latest versions of all files, you can enter:

```
   ?DELETE (*(+))
```

STEP 6: Enter the SCUP COMPRESS command to compress the code file.

In this case, COMPRESS reclaims the disk space used by the deleted programs by compressing the code file "pobjcod":

```
   ?COMPRESS pobj
```

SCUP renames the old object file "$data.enab.aa777cod" (for recovery purposes) and then purges "$data.enab.aa777cod" when compression is complete:

```
   COMPRESS $DATA.ENAB.POBJ
   EMPLOYEE-PROG (4)                    EMPLOYEE-PROG (1)
   OLD FILE $DATA.ENAB.POBJ RENAMED TO $DATA.ENAB.AA777
   COMPRESSED $DATA.ENAB.POBJ
```

Figure 10-1. Using SCUP to Reclaim Disc Space (Continued)

SECTION 11

INTEGRATING APPLICATIONS INTO A SINGLE PATHWAY SYSTEM

When you generate several ENABLE applications, you often must establish a separate PATHWAY system for each, and execute each individually. If you use these applications on a regular basis, you may want to establish a single PATHWAY system for them all. If you often use information from one application in conjunction with another application, you may want to provide the ability to call the second application from the first.

When you integrate applications into a single PATHWAY system with one PATHMON process and one TCP, the TCP can use the object code from any of the applications to send requests to the General Server.

Figure 11-1 illustrates a PATHWAY system with several integrated applications. A user of this PATHWAY system can call "prog-a," "prog-b," and "prog-c" from a menu.

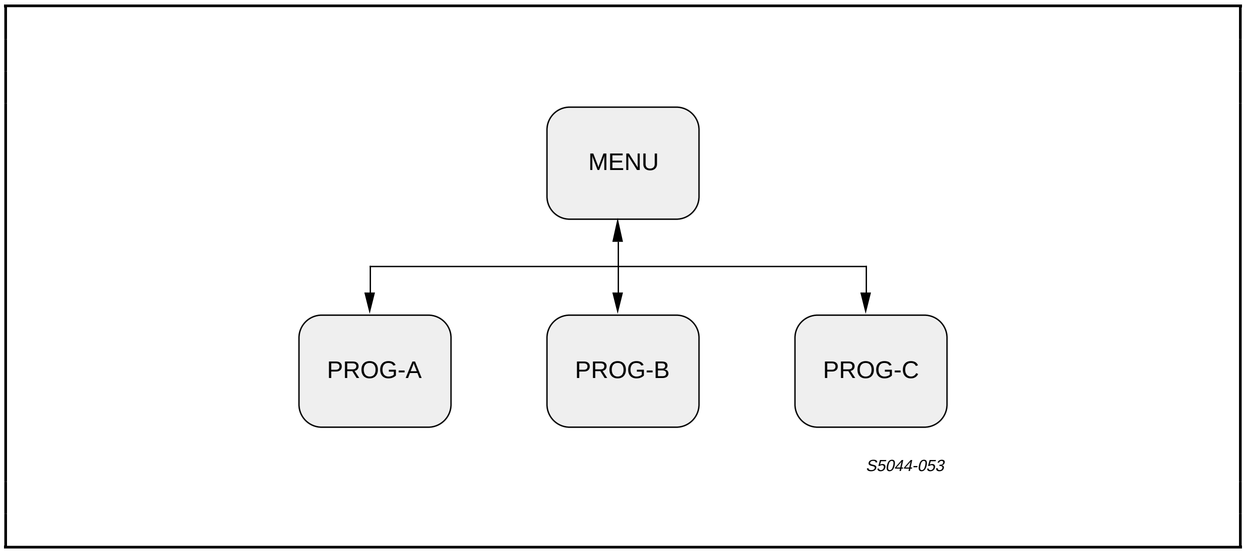


Figure 11-1. Common PATHWAY System

In addition to the usual organization of applications shown in Figure 11-1, another type of organization is possible. Figure 11-2 illustrates a PATHWAY system where the user can call "prog-b" from "prog-a" and "prog-c" from "prog-b."

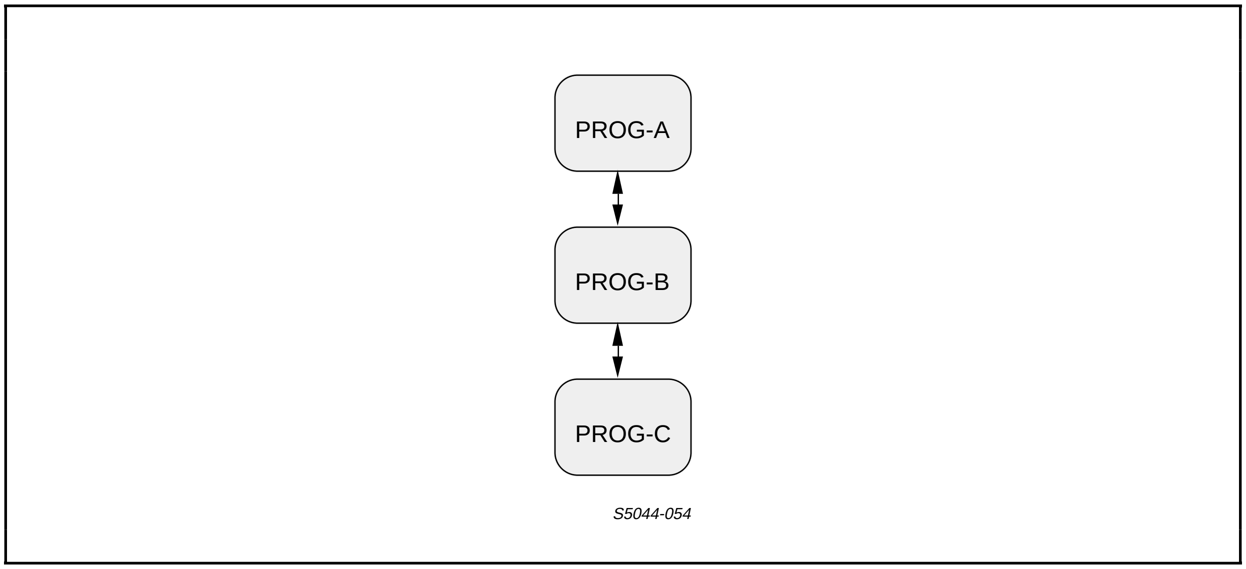


Figure 11-2. Alternative PATHWAY System

This section discusses the tasks involved in integrating applications into a single PATHWAY system, including:

1. Writing a SCREEN COBOL menu program that you can use to call the integrated applications

2. Generating the applications in such a manner that they can be integrated into the same PATHWAY system

3. Optionally, modifying the SCREEN COBOL source code of an application generated by ENABLE to include a call to another application

4. Modifying a PATHCOM command file to include the PATHWAY entities needed to execute all the applications

Section 12 describes several applications that are integrated into a single project-tracking system.

WRITING A SCREEN COBOL MENU PROGRAM

A SCREEN COBOL menu program provides you with a method of selecting the application you want to use. When you write a SCREEN COBOL menu program, you must include program logic that:

- Defines a menu screen

- Displays the menu screen

- Accepts information from the menu screen and uses this information to determine which ENABLE application is to be called

- Calls the appropriate ENABLE application

Optionally, you can also include program logic that limits access to all or part of the PATHWAY system.

Refer to Section 12 for the source code of a sample menu program.

GENERATING THE APPLICATIONS

When you generate the applications, you can simplify the process of integrating them by performing the following tasks:

1. Direct all SCREEN COBOL object code to the same object file. When the object code for all the applications resides on the same file, you eliminate the need to modify a portion of the PATHCOM command file.

2. Obtain a SCREEN COBOL compilation listing for each application. These compilation listings provide information that you will need when you modify the PATHCOM command file.

3. Obtain SCREEN COBOL source code for each application that is to call another. Since ENABLE does not generate applications that call other applications, you must modify the source code of the calling application if you want to provide this capability.

4. Avoid using the same box names for boxes having different values for the ABILITY attributes, or for boxes that represent different data base files. Otherwise, you may not be able to integrate the applications into a single PATHWAY system.

Directing Object Code to the Same Object File

When you use an ENABLE application, a terminal control process (TCP) within the PATHWAY system interprets the SCREEN COBOL object code and sends messages to a server process. A TCP can obtain SCREEN COBOL object code from several different object files. If you direct the object code of all applications to a single object file, however you eliminate the need to identify these different object files in the PATHCOM command file.

To direct object code to the same object file, you can use either of the following methods:

1. Generate the applications on the same system, volume, and subvolume.

2. Supply a value for the SCOBOLOBJECT attribute to identify the same object file for all applications.

Generating the Applications on the Same System, Volume, and Subvolume

If you do not supply a value for the SCOBOLOBJECT attribute, the SCREEN COBOL compiler (SCOBOLX) directs the object code to a file named "pobjcod" on the default system, volume, and subvolume. The compiler also writes a directory entry for this object code to a file named "pobjdir" on the default system, volume, and subvolume.

If you do not define new defaults by using the ENABLE operating commands, the default system, volume, and subvolume are those in effect when you enter ENABLE.

Supplying a Value for the SCOBOLOBJECT Attribute

If you generate your applications on different systems, volumes, or subvolumes, supply the fully expanded name of the object file as a value for the SCOBOLOBJECT attribute. For example, the following SET APPL SCOBOLOBJECT command identifies a file named "\xyz.$mkt.sample.aobjcod" as the file to which the object code is to be directed:

```
   SET APPL SCOBOLOBJECT \xyz.$mkt.sample.aobj
```

To obtain the name of the object file, the SCREEN COBOL compiler appends the characters COD to the value you supply. Refer to the ENABLE Reference Manual for more information about expanded file names.

Obtaining a SCREEN COBOL Compilation Listing

For each application, a SCREEN COBOL compilation listing provides you with information needed to modify the PATHCOM command file. To obtain a compilation listing, supply the name of the listing file as a value for the SCOBOLLIST attribute. For example, the following SET APPL SCOBOLLIST command indicates that ENABLE is to write the SCREEN COBOL compilation listing to a file named "lista":

```
   SET APPL SCOBOLLIST lista
```

Identify a different listing file for each application.

Obtaining SCREEN COBOL Source Code

If you want to provide the ability to call one application from another, you will need the SCREEN COBOL source code of the calling application. To obtain the SCREEN COBOL source code, supply a value for the SCOBOLSOURCE attribute, which identifies a file to which the SCREEN COBOL source code is written. The following SET APPL SCOBOLSOURCE command, for example, indicates that the SCREEN COBOL source code is written to a file named "srca":

```
   SET APPL SCOBOLSOURCE srca
```

Avoiding Conflicting Box Names

When you integrate several applications into the same PATHWAY system, some applications may use boxes having the same name. Since box names appear both in the generated SCREEN COBOL program and in the PATHCOM command file, all boxes with the same name must:

- Represent the same data base file
- Have compatible values for the ABILITY attributes (FILL, DELETE, INSERT, READ, and UPDATE)

ENABLE uses the box name in a SET SERVER ASSIGN command in the PATHCOM command file to associate a data base file with the named box. The SET SERVER ASSIGN command for an application that uses a box named "one-box," for example, might appear as follows:

```
   SET SERVER (ASSIGN ONE-BOX, ONEFILE, INPUT)
```

where "one-box" is the box name, "onefile" is the name of the data base file that the box represents, and INPUT is the access mode with which the file is to be opened.

When you establish a PATHWAY system, PATHMON, the controlling process in the system, stores information from the SET SERVER ASSIGN commands in a file called PATHCTL. When you execute the application, the following events take place:

- PATHMON starts the General Server and passes it information from the SET SERVER ASSIGN and SET SERVER PARAM commands.

- The SCREEN COBOL requester program sends a message to the General Server that tells it to open a file. The requester program does not use the file name, but uses the box name instead.
- The General Server accepts the message from the requester program and looks at the assign specification to determine which physical file is assigned to a specific box name.
- The General Server then opens the assigned physical files, using the access mode indicated in the assign specification.

Figure 11-3 illustrates this process.

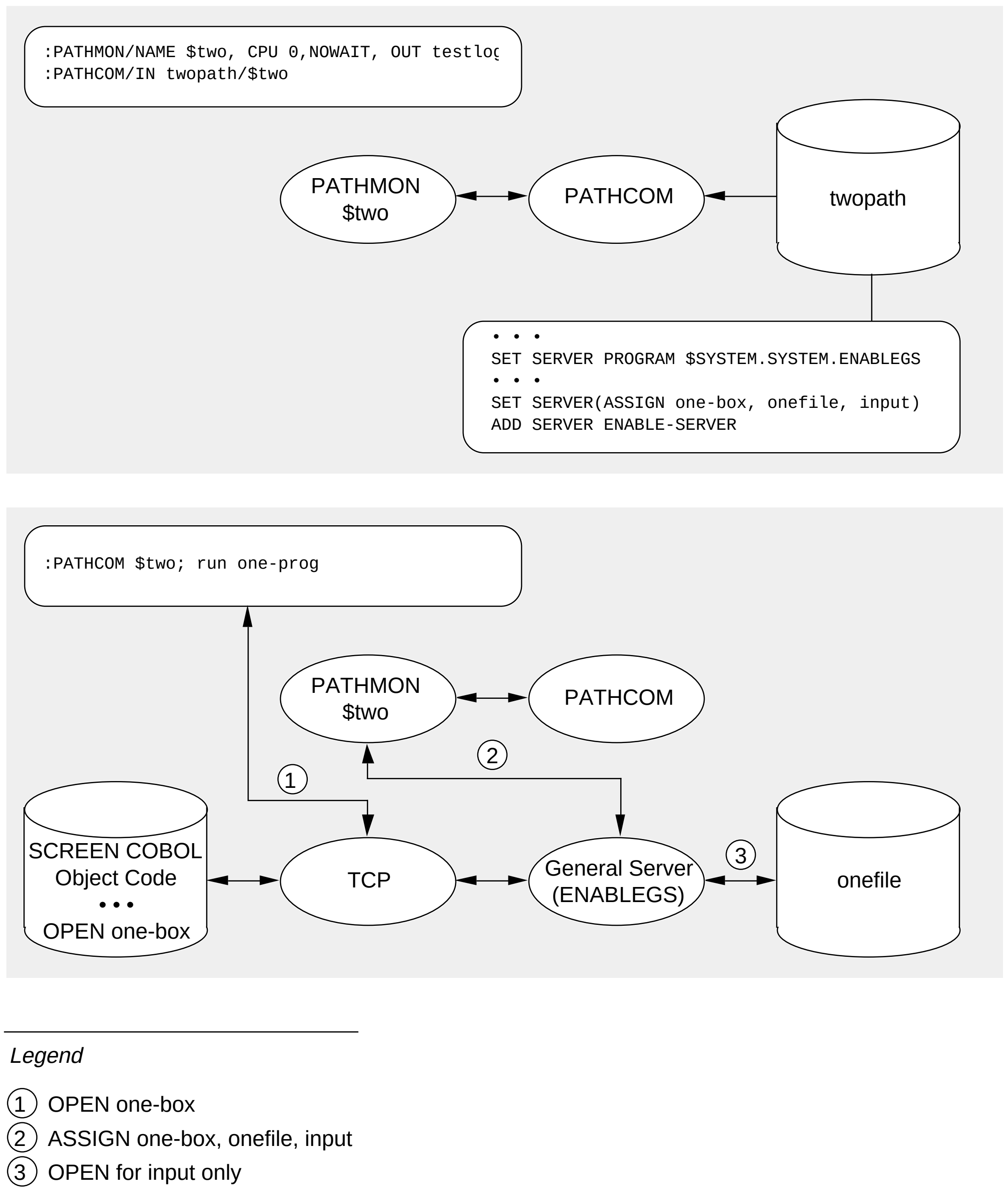

:PATHMON/NAME $two, CPU 0,NOWAIT, OUT testlog
:PATHCOM/IN twopath/$two
PATHMON $two
PATHCOM
twopath
• • •
SET SERVER PROGRAM $SYSTEM.SYSTEM.ENABLEGS
• • •
SET SERVER(ASSIGN one-box, onefile, input)
ADD SERVER ENABLE-SERVER
:PATHCOM $two; run one-prog
PATHMON $two
PATHCOM
1
2
3
SCREEN COBOL Object Code • • • OPEN one-box
TCP
General Server (ENABLEGS)
onefile
Legend
1 OPEN one-box
2 ASSIGN one-box, onefile, input
3 OPEN for input only
S5044-055

Figure 11-3. Box Names in a PATHWAY System

Since the General Server can assign only one physical file to a box name, boxes with the same name must represent the same file. Consider, for example, the following ENABLE commands used to generate two applications:

```
   SET RECORD a-rec             SET RECORD b-rec
   SET DATAFILE afile           SET DATAFILE bfile
   ADD BOX my-box               ADD BOX my-box
   SET APPL PATHCOMFILE p1      SET APPL PATHCOMFILE p2
   SET APPL TREE (01 my-box)    SET APPL TREE (01 my-box)
   ADD APPL my-appl1            ADD APPL my-appl2
   GENERATE APPL my-appl1       GENERATE APPL my-appl2
```

You cannot integrate these applications into the same PATHWAY system because "my-box" in "my-appl1" represents a data base file named "afile" while "my-box" in "my-appl2" represents a data base file named "bfile."

Since the values of the ABILITY attributes determine the access mode with which the General Server can open a data base file, boxes with the same name must also have compatible values for these attributes. The General Server cannot assign a file to box when the file is assigned more than one access mode.

Table 11-1 lists the access modes and the values for the ABILITY attributes that correspond to these access modes.

Table 11-1. Access Modes and ABILITY Attribute Values

Access Mode	Ability Attribute Value
I/O (default)	READ ON (BOXFILL can be either ON or OFF) DELETE, UPDATE, or INSERT set ON
Input only	READ ON or BOXFILL ON DELETE OFF, INSERT OFF, and UPDATE OFF
Output only	READ OFF and BOXFILL OFF INSERT ON DELETE OFF and UPDATE OFF S5044-056

If boxes with the same name have different access modes, you may not be able to use the applications in the way that you expect. For example, suppose that you use the following commands to generate two applications:

```
   SET RECORD c-rec                SET RECORD c-rec
   SET DATAFILE cfile              SET DATAFILE cfile
   SET DELETE OFF
   SET UPDATE OFF
   SET INSERT OFF
   ADD BOX new-box                 ADD BOX new-box
   SET APPL PATHCOMFILE p4         SET APPL PATHCOMFILE p3
   SET APPL TREE (01 new-box)      SET APPL TREE (01 new-box)
   ADD APPL new-appl2              ADD APPL new-appl1
   GENERATE new-appl2              GENERATE APPL new-appl1
```

The SET SERVER ASSIGN commands for these applications would appear as follows:

```
   SET SERVER ASSIGN (new-box, cfile, INPUT)    (for new-appl1)

   SET SERVER ASSIGN (new-box, cfile)           (for new-appl2)
```

If you enter both of these commands in the same PATHCOM command file and then establish a PATHWAY system with that command file, the General Server uses the access mode of the SET SERVER ASSIGN command that appears last in the command file. If the SET SERVER ASSIGN command for "new-app2" appears last, both applications will work as expected. If the SET SERVER ASSIGN command for "newappl1" appears last, "newappl2" will not work as expected; that is, you will not be able to use this application to delete, insert, or update records because the access mode (INPUT) with which the General Server will open the file will not permit these operations.

If you plan to integrate your applications into a single PATHWAY system, you can avoid using conflicting box names by doing the following:

1. Describe, name, and add the boxes necessary for an application.

2. Add the application to the object table.

3. Repeat steps 1 and 2 until you have named and added every application to be integrated.

4. Use the GENERATE * command to generate all of the applications.

Since ENABLE will not allow you to add an object if an object with that name already exists, this procedure prevents you from using conflicting box names.

MODIFYING SCREEN COBOL SOURCE CODE TO CALL ANOTHER APPLICATION

To provide the capability to call one application from another, you must modify the SCREEN COBOL source code of the calling application.

Suppose that you have an application, "A," that displays part of a record, and another application, "AA," that displays the entire record. In certain circumstances, the user may want to press a function key (shifted F3 is reserved for this purpose) to go directly from the partial record display to a display of the entire record.

When ENABLE generates the SCREEN COBOL source code for an application, it includes a paragraph named T9155-CHAIN that contains code executed by the application when a user presses the shifted F3 function key. Normally, this code causes the application to display a message stating that the function key is not supported. You can, however, modify this paragraph to include code that causes a call to another application, by performing the following tasks:

1. Use the text editor to edit the SCREEN COBOL source code for the calling application. If the source code for an application is in a file named "applscr," for example, you can edit this file by entering:

   ```
   :EDIT applsrc
   ```

2. Find the T9155-CHAIN paragraph in the SCREEN COBOL source code by entering the following:

   ```
   *LIST /T9155-CHAIN./A
   ```

   When you enter this command, the editor returns the number of the line that contains the T9155-CHAIN paragraph.

3. Use the editor to examine the T9155-CHAIN paragraph, for example:

```
*LIST 2775/2785

  2775  T9155-CHAIN.
  2776    IF T9155-AAABOX-CHOICE
  2777      MOVE T9155-NOT-SUPPORTED-MESSAGE TO T9155-ERROR-MSG
  2778    ELSE IF T9155-ABBBOX-CHOICE
  2779      MOVE T9155-NOT-SUPPORTED-MESSAGE TO T9155-ERROR-MSG
  2780    ELSE NEXT SENTENCE.
  2781  ...
*
```

   Note that the source code that appears between the phrase T9155-CHAIN and the clause ELSE NEXT SENTENCE is the only source code that affects the ability of the application to call another application.

4. Use a SCREEN COBOL CALL statement to replace the "MOVE T9155-NOT-SUPPORTED-MESSAGE TO T9155-ERROR-MSG" sentence. In its simplest form a CALL statement appears as follows:

```
CALL <application-name>
```

   Note the the T9155-CHAIN paragraph might refer to more than one box. In the preceding example, this paragraph refers to two boxes:

   - "aaa" (T9155-AAABOX-CHOICE)
   - "abb" (T9155-ABBBOX-CHOICE)

   To call an application named "AA" from the box named "aaa," replace the MOVE statement for this box with a CALL statement, for example:

```
2775  T9155-CHAIN.
2776    IF T9155-AAABOX-CHOICE
2777      CALL "AA"
2778    ELSE IF T9155-ABBBOX-CHOICE
2779      MOVE T9155-NOT-SUPPORTED-MESSAGE TO T9155-ERROR-MSG
2780    ELSE NEXT SENTENCE.
```

You can then replace the MOVE statement for the "abb" box with the T9155-CANT-CHAIN-MESSAGE provided by ENABLE, for example:

```
T9155-CHAIN.
  IF T9155-AAABOX-CHOICE
    CALL "AA"
  ELSE IF T9155-ABBBOX-CHOICE
    MOVE T9155-CANT-CHAIN-MESSAGE TO T9155-ERROR-MSG
  ELSE NEXT SENTENCE.
```

When you replace a MOVE statement with the T9155-CANT-CHAIN-MESSAGE, the application displays a message whenever a user presses shifted F3 without first positioning the cursor within the appropriate box.

Alternatively, you can allow a user to call an application from both "aaa" box and "abb" box by replacing the MOVE statements for both boxes with appropriate CALL statements. If you replace all of the MOVE statements within the T9155-CHAIN paragraph with CALL statements, the user can call another application without positioning the cursor within a specific box.

5. Compile the modified source code by using the SCOBOLX run command:

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

   <source-file-name>

   is the name of the file containing the source code.

   <list-file-name>

   is the name of the file to which the SCREEN COBOL listing is to be written.

   <object-file-name>

   is the name of the file for the object code; this file should be the same as the object file for the other applications.

6. Keep the compilation listing of the modified SCREEN COBOL program. You will need this listing when you modify the PATHCOM command file.

MODIFYING A PATHCOM COMMAND FILE

To establish a PATHWAY system that integrates several ENABLE applications, you can modify the PATHCOM command file generated for any one of them. The PATHCOM command file generated for a single application contains the PATHWAY commands required to execute that application. To include the PATHWAY commands that allow execution of more than one application, select a PATHCOM command file and modify it using the text editor as follows:

1. Increase the value specified for MAXTERMDATA to include enough data space for the applications.

2. Change the program name specified for SET PROGRAM TYPE and ADD PROGRAM to the name of your menu program.

3. Include all the appropriate SET SERVER ASSIGN commands from the other PATHCOM command files.

4. Optionally, include additional commands that affect the PATHWAY system, such as commands that define and control terminals.

Figure 11-4 shows the commands that must be modified in a sample PATHCOM command file. (The application used to generate this file is shown in Section 12.)

```
SET PATHMON BACKUPCPU 1
SET PATHWAY MAXTCPS 10
SET PATHWAY MAXTERMS 10
SET PATHWAY MAXPROGRAMS 10
SET PATHWAY MAXSERVERCLASSES 10
SET PATHWAY MAXSERVERPROCESSES 10
SET PATHWAY MAXSTARTUPS 10
SET PATHWAY MAXPATHCOMS 40
SET PATHWAY MAXASSIGNS 32
SET PATHWAY MAXPARAMS 32
START PATHWAY COLD!

SET TCP PROGRAM $SYSTEM.SYSTEM.PATHTCP2
SET TCP CPUS 0:1
SET TCP MAXTERMS 5
SET TCP MAXSERVERCLASSES 003
SET TCP MAXSERVERPROCESSES 003
SET TCP MAXTERMDATA 12036  <---  increase MAXTERMDATA
SET TCP MAXREPLY    02000        value to provide more data
SET TCP NONSTOP 0                space
SET TCP TCLPROG  $DATA.SAMPLE.POBJ
ADD TCP ENABLE-TCP

SET PROGRAM TCP ENABLE-TCP
SET PROGRAM TYPE T16-6520 INITIAL APPL-A  <-  change
SET PROGRAM TMF OFF                           initial name
ADD PROGRAM APPL-A  <-----------------------  to menu
                                              program
RESET SERVER ASSIGN, PARAM

SET SERVER PROGRAM $SYSTEM.SYSTEM.ENABLEGS
SET SERVER CPUS 0:1
SET SERVER NUMSTATIC 1
SET SERVER (ASSIGN A-BOX,AFILE,INPUT)
SET SERVER (ASSIGN B-BOX,BFILE)
SET SERVER (ASSIGN C-BOX,CFILE)  <-  add server assignments
SET SERVER TMF OFF                   for other applications
ADD SERVER ENABLE-SERVER
```

Figure 11-4. Sample PATHCOM Command File

Determining a New Value for MAXTERMDATA

The value of MAXTERMDATA defines the maximum number of bytes used by the terminal control process (TCP) for data space for each terminal. A PATHCOM command file generated by ENABLE contains a value for MAXTERMDATA appropriate for one application. For example, in Figure 11-4, MAXTERMDATA is 12036.

When you integrate several ENABLE applications, you must increase the value of MAXTERMDATA to allow the necessary data space for all the applications. You can estimate an appropriate value for MAXTERMDATA following these steps:

1. Examine the SCREEN COBOL listings of all the applications. On the last page of each listing, the SCOBOL compiler prints a value called DATA SIZE. Make a list of the DATA SIZE values for each application.

2. Compute the number of bytes in the longest path through the applications. A path through applications exists when one application calls another application. Figure 11-5 shows two examples of how to compute the number of bytes in the longest path. The first example shows how you determine the longest path when all applications are called from a menu program. The second example shows how you determine the longest path when applications are called from other applications.

3. Multiply the number of bytes in the longest path by 2 to estimate an appropriate value for MAXTERMDATA, as follows:

    For Example 1 in Figure 11-5 (where all applications are called from a menu program), the estimated value of MAXTERMDATA is 15048 (7524 * 2).

    For Example 2 in Figure 11-5 (where applications are called from other applications), the estimated value of MAXTERMDATA is 21628 (10818 * 2).

Refer to the PATHWAY System Management Reference Manual for more information about MAXTERMDATA.

Changing the Program Name in the PATHCOM SET PROGRAM TYPE Command

The SET PROGRAM TYPE command identifies the type of terminal upon which a SCREEN COBOL program will execute. This command also identifies the SCREEN COBOL program unit that the terminal enters on startup. In Figure 11-4, this command appears as follows:

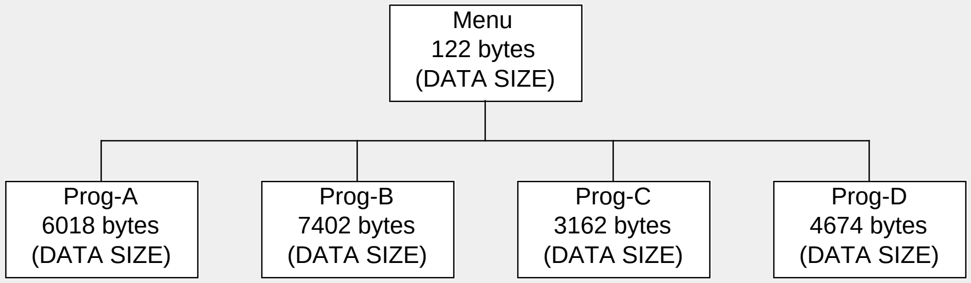
```
SET PROGRAM TYPE T16-6520 INITIAL APPL-A
```

where T16-6520 is the type of terminal upon which the application will execute and APPL-A is the SCREEN COBOL program name.

To integrate several ENABLE applications and use a menu program, change the program name to that of the SCREEN COBOL menu program (the name that appears in the PROGRAM-ID paragraph of the menu program). If the name of your menu program is MENU, for example, you would change the program name to:

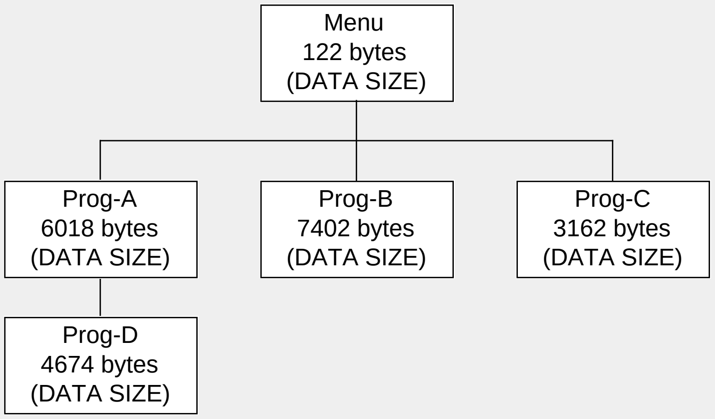
```
SET PROGRAM TYPE T16-6520 INITIAL MENU
```

Example 1



```
Menu to Prog-A  =  122 bytes + 6018  =  6140 bytes
Menu to Prog-B  =  122 bytes + 7402  =  7524 bytes
Menu to Prog-C  =  122 bytes + 3162  =  3284 bytes
Menu to Prog-D  =  122 bytes + 4674  =  4796 bytes
Longest Path    =    Menu to Prog-B  =  7524 bytes
```

Example 2



```
Menu to Prog-A to Prog-D  =  122 bytes + 6018 bytes
                                       + 4674 bytes  =  10814 bytes
Menu to Prog-B            =  122 bytes + 7402        =   7524 bytes
Menu to Prog-C            =  122 bytes + 3162        =   3284 bytes
Longest Path      =      Menu to Prog-A to Prog-D    =  10818 bytes
```

S5044-057

Figure 11-5. Determining Longest Path

Changing the Program Name in the PATHCOM ADD PROGRAM Command

The ADD PROGRAM command enters a program description into the PATHWAY configuration. A PATHCOM command file generated by ENABLE sets the program-name parameter of this command to the application name. When you integrate several ENABLE applications, change the program name to that of the SCREEN COBOL menu program. (This name appears in the PROGRAM-ID paragraph of the menu program.) In Figure 11-4, for example, the ADD PROGRAM command appears as follows:

```
   ADD PROGRAM APPL-A
```

If your menu program is named MENU, change the program name as follows:

```
   ADD PROGRAM MENU
```

Adding SET SERVER ASSIGN Commands

The General Server obtains file assignments from the SET SERVER ASSIGN commands in the PATHCOM command file. A PATHCOM command file generated by ENABLE contains file assignments for a single ENABLE application. In Figure 11-4, for example, the following SET SERVER ASSIGN commands appear:

```
   SET SERVER (ASSIGN A-BOX,AFILE,INPUT)
   SET SERVER (ASSIGN B-BOX,BFILE)
   SET SERVER (ASSIGN C-BOX,CFILE)
```

When you integrate several ENABLE applications, you must add the SET SERVER ASSIGN commands needed for all of the applications. You can copy the SET SERVER ASSIGN commands from the PATHCOM command files produced for the applications to be added, for example:

```
   SET SERVER (ASSIGN A-BOX,AFILE,INPUT)
   SET SERVER (ASSIGN B-BOX,BFILE)  <------  original SET
   SET SERVER (ASSIGN C-BOX,CFILE)           SERVER ASSIGN
   SET SERVER (ASSIGN CC-BOX, CFILE)         commands

   SET SERVER (ASSIGN D-BOX, DFILE)  <-----  SET SERVER ASSIGN
   SET SERVER (ASSIGN E-BOX, EFILE, INPUT)   commands copied
   SET SERVER (ASSIGN BB-BOX, BFILE, INPUT)  from other
   SET SERVER (ASSIGN AA-BOX, AFILE)         PATHCOM files
   ...
```

If some applications access files audited by TMF and other applications access nonaudited files, you must use different copies of the General Server for the audited and nonaudited files. In this case, the PATHCOM command file must contain at least two sets of SERVER commands, for example:

```
   SET SERVER PROGRAM $SYSTEM.SYSTEM.ENABLEGS
   SET SERVER CPUS 0:1
   SET SERVER NUMSTATIC 1  <----------------  commands that
   SET SERVER (ASSIGN A-BOX,AFILE,INPUT)      define a server
   SET SERVER (ASSIGN B-BOX,BFILE)            to access non-
   SET SERVER (ASSIGN C-BOX,CFILE)            audited files
   SET SERVER TMF OFF
   ADD SERVER ENABLE-SERVER

   SET SERVER PROGRAM $SYSTEM.SYSTEM.ENABLEGS
   SET SERVER CPUS 0:1
   SET SERVER NUMSTATIC 1  <----------------  commands that
   SET SERVER (ASSIGN P-BOX, PFILE)           define a server
   SET SERVER TMF ON                          to access non-
   SET SERVER (PARAM TMF ON)                  audited files
   ADD SERVER TMF-SERVER
```

If your applications access audited and nonaudited files, be sure that you add the SET SERVER ASSIGN commands to the correct copy of the General Server. The SET SERVER ASSIGN commands for audited files must be included with the commands for the server that has TMF ON. The SET SERVER ASSIGN commands for nonaudited files must be included with the commands for the server that has TMF OFF.

Including Optional Commands

You can make other changes and additions to the PATHCOM command file to define the PATHWAY system further. For example, you can prohibit execution of the applications unless the General Server can open all the data base files by adding the following command:

```
   SET SERVER (PARAMS ALLFILES ON)
```

You can also add commands that define and control terminals of the terminal type determined when the applications were generated. To do this, add the following command for each terminal defined for the PATHWAY system:

- SET TERM FILE filename

  This command defines the system name of a terminal to be added to the PATHWAY system. Set the value of filename to the name of the terminal.

- SET TERM INITIAL program-unit-name

  This command defines the SCREEN COBOL program unit that the terminal enters on start-up. Set program-unit-name to the name of the SCREEN COBOL menu program appropriate for the terminals. (This is the same as the program name following the INITIAL specification in the SET PROGRAM TYPE command.)

- SET TERM tcp

  This command defines the TCP that controls the terminal. Generally, you can set the value of tcp to ENABLE-TCP.

- ADD TERM termname

  This command enters a description of a terminal into the PATHWAY system. In this command, termname can be the name of the terminal. The name must begin with a letter and can contain 1 through 15 alphanumeric characters or hyphens. The name must be unique within the PATHWAY system.

The PATHCOM file generated by ENABLE allows you to define up to five terminals per TCP. (The number of terminals that can be defined is determined by the value established by the SET PATHWAY MAXTERMS command and the value established by the SET TCP MAXTERMS command.)

After you establish your PATHWAY system, you can use separate obey files to start and stop the appropriate terminals. Figure 11-6 shows a sample obey file that starts a TCP and a terminal named "term01."

```
PATHCOM $one; START TCP ENABLE-TCP; START TERM term01
```

Figure 11-6. Sample Obey File That Starts a Terminal

Figure 11-7 shows a sample obey file that stops the PATHWAY system which "term01" is using.

```
PATHCOM $one; SHUTDOWN, WAIT
```

Figure 11-7. Sample Obey File That Stops a PATHWAY System

Refer to the PATHWAY System Management Reference Manual for more information about adding terminals to a PATHWAY system.

INTEGRATING AN APPLICATION INTO AN EXISTING PATHWAY SYSTEM

You can add an ENABLE application to an existing PATHWAY system by performing the tasks described in the following paragraphs. Some of these tasks apply to application generation; other tasks apply to application integration.

Generating the Application

When you generate an application for integration within an existing PATHWAY system:

1. Direct the object code for the generated application to an object file used by the existing PATHWAY system. To perform this task, supply the name of the object file as a value for the SCOBOLOBJECT attribute. For more information, refer to the discussion "Directing Object Code to the Same Object File" earlier in this section.

2. Request a PATHCOM command file. You can use information from this file when you integrate the application into the existing PATHWAY system. To request a PATHCOM command file, supply a file name as the value of the PATHCOMFILE attribute. Refer to Sections 4 and 5 for more information.

Integrating the Application

To integrate an application into an existing PATHWAY system, use the INFO PATHWAY, INFO TCP *, INFO PROGRAM * and INFO SERVER * commands to become familiar with the existing configuration. You can then use the following PATHCOM commands to modify this configuration. Refer to the PATHWAY System Management Reference Manual for more information about these commands.

- SET PATHWAY MAXASSIGNS

  This command defines the maximum number of ASSIGN specifications that can be entered into the PATHWAY configuration for each server class. The value of this parameter must be large enough to allow for the existing ASSIGN specifications and the additional ASSIGN specifications that you must add for the generated application. If the value of SET PATHWAY MAXASSIGNS is not large enough, you must SHUTDOWN and reconfigure the PATHWAY system before integrating the new application.

- SET PATHWAY MAXPARAMS

  This command defines the maximum number of PARAM specifications that can be entered into the PATHWAY system for all server classes. The value of this parameter must be large enough to allow for the existing PARAM specifications and any additional PARAM specifications you must add for the generated application.

- SET PATHWAY MAXSERVERCLASSES

  This command defines the maximum number of server class descriptions that can be entered into the PATHWAY configuration. If you must add a description of the General Server this value must be large enough to allow for all existing server-class descriptions and the server-class description for the General Server.

- SET TCP MAXREPLY

  This command identifies the maximum number of bytes permitted in a reply from a server. The value of this parameter must be equal to or greater than the value of MAXREPLY in the generated PATHCOM command file.

- SET TCP MAXTERMDATA

  This command identifies the number of bytes used for data area for each terminal. The value of this parameter must be equal to or greater than the value of MAXTERMDATA in the generated PATHCOM command file.

- SET TCP MAXSERVERCLASSES

  This command identifies the maximum number of server classes with which the TCP can establish links. If you must add a server class description for the General Server, the value of this parameter must be large enough to allow for the existing server classes and the new description of the General Server server class.

  If the existing system configuration already defines a server class for the General Server, use the PATHCOM ALTER command to add appropriate SET SERVER ASSIGN commands for the new application. When you use the ALTER command, be sure to copy the SET SERVER ASSIGN commands from the generated PATHCOM command file.

  If the existing configuration does not define a server class for the General Server, use the PATHCOM SET and ADD commands to add the appropriate definition.

SECTION 12

SAMPLE PROJECT-TRACKING SYSTEM

This section describes the tasks involved in developing a sample project-tracking system, consisting of several ENABLE applications integrated into a single PATHWAY system. Figure 12-1 illustrates this sample system.

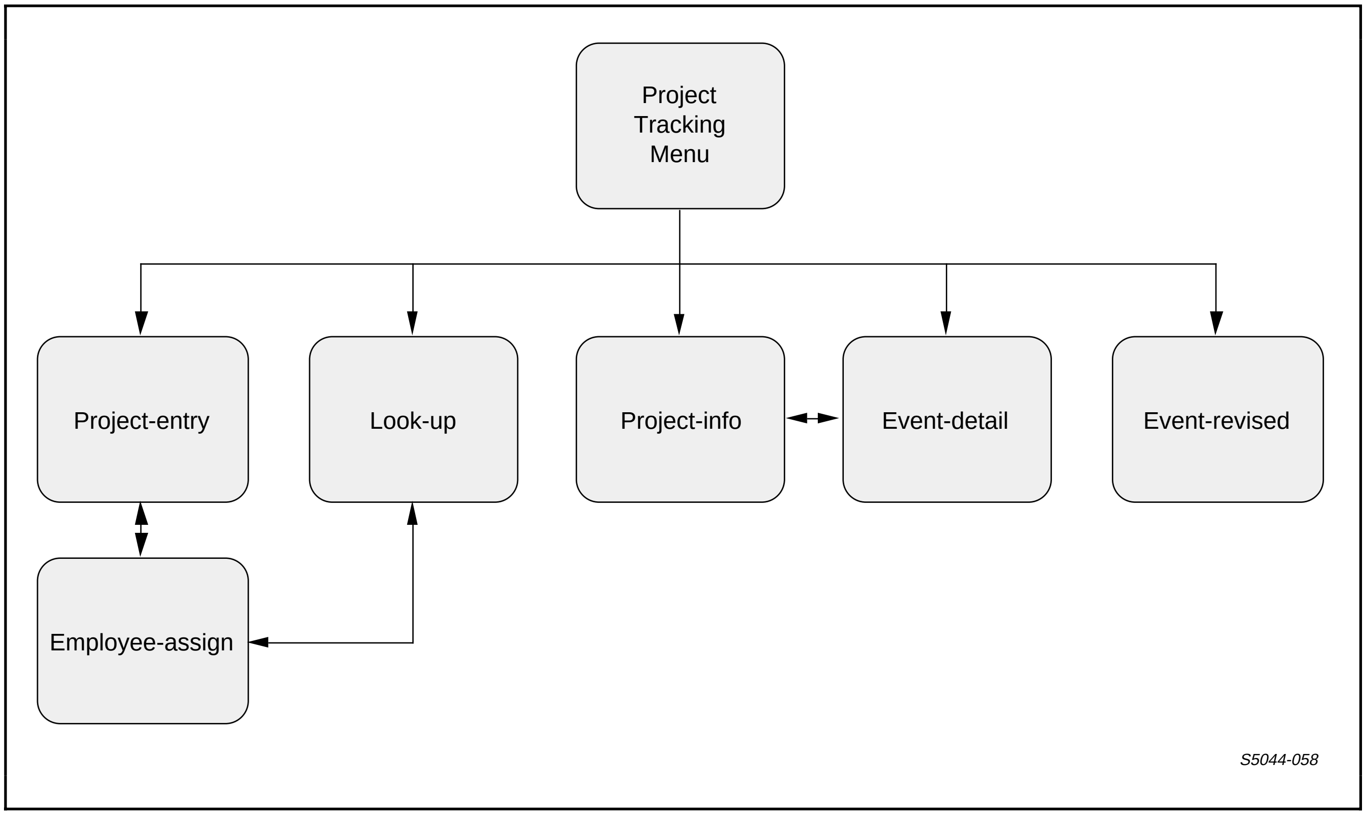


Figure 12-1. Sample Project-Tracking System

The tasks involved in developing this system are:

1. Defining the functional requirements of the system

2. Defining the data requirements of the system

3. Defining the applications to be generated

4. Generating the applications

5. Modifying the SCREEN COBOL source code of selected applications

6. Writing a SCREEN COBOL Menu program

7. Modifying a PATHCOM command file to integrate the applications

DEFINING THE FUNCTIONAL REQUIREMENTS OF THE SYSTEM

The first step in developing an application system is to define the functions it provides, based on requests from potential users. By eliciting the cooperation of potential users, you can find out about the tasks they perform and how the system can help them.

Suppose that discussions with potential users indicate that the system should provide the following functions:

1. A method for entering information about new projects and the events associated with each

2. A means for assigning an employee to a specific event

3. A means for displaying general information about each project and its events

4. A means for displaying detailed information about a specific event within a project

5. An allowance for revision of certain information associated with an event

DEFINING THE DATA REQUIREMENTS OF THE SYSTEM

The next step in developing an application system is to identify the information needed to carry out each function. To define these data:

1. Identify classes of data that interest the users of the application system. These classes of data will correspond to files accessed by the application system.

2. Identify the data items that belong to each class of data. These data items will correspond to the fields within the files.

3. Determine the relationships that exist between classes. Identifying these relationships will help you later when you build the tree structures for the applications.

4. List the fields in each file. This step and the following step will help you when you create record descriptions for the files.

5. Identify key fields for each file.

6. Normalize the files. Normalized files are simple to maintain, easy to link within applications, and easy to obtain reports from when using ENFORM.

7. Use the Data Definition Language (DDL) to create the record descriptions and data dictionary that describe the files.

Identifying Classes of Data

For the project-tracking system, you can determine two classes of data that are of interest to users: the projects that application tracks, and the events associated with each project.

You can identify these classes more precisely by examining the way that projects are currently tracked within the department, and through discussion with its employees, you might discover the following procedure:

1. A project manager fills out a form that describes the project, listing a project code (a unique number that identifies each project), a brief description of the project and its estimated starting and ending dates. The project manager then files this form for later use.

2. Next, the project manager fills out forms for each event. Each form describes an event, its projected starting and ending dates, and the names of the employees assigned to it. The project manager files the event forms in some convenient order.

3. The project manager makes copies of each event form and passes them to the pertinent employees.

4. If an employee needs to revise the projected starting or ending date, he or she fills out an event-revision form. The employee keeps a copy of this form and gives the original to the project manager.

5. To record the current status of a project, the project manager assembles the original project form, the event forms, and any event-revision forms. The project manager then fills out another status-summary form for the project.

6. The department secretary collects all of the forms and produces a monthly report.

By using the preceding information, you could determine that the department uses the classes of data shown in Figure 12-2.

PROJECTS	The projects being tracked within your department
EVENTS	The events associated with each project
EMPLOYS	The employees assigned to each event

Figure 12-2. Classes of Data Within the Project-Tracking System

Identifying the Data Items

Next, data items must be identified and associated with each class. For example, specific data items of the project-tracking system can be associated with each class as shown in Figure 12-3.

PROJECTS

project-code	--a unique number that identifies the project
project-desc	--a brief description of the project
proj-status	--the current status of the project
proj-start-date	--the projected starting date
proj-end-date	--the projected ending date
proj-mgr	--the employee number of the project manager
manager-name	--the name of the project manager
manager-dept	--the department number of the project manager

EVENTS

event-num	--a number that identifies the event within the project
event-desc	--a brief description of the event
predict-starting-date	--the predicted starting date
predict-ending-date	--the predicted ending date
revised-starting-date	--the revised starting date
revised-ending-date	--the revised ending date
change-info	--a brief description of the reason for the date revision
update-emp	--a number that identifies an employee who made a date revision
emp-name (...)	--the names of the employees assigned to an event

EMPLOYS

emp-no	--a unique number that identifies an employee
emp-name	--the name of the employee
emp-dept	--a number that identifies the department number of the employee

The symbol (...) indicates that the item may be repeated.

Figure 12-3. Data Items Associated With Each Class of Data

After they have been identified, you can ask specific questions about each data item; for example:

- Are any of the data items associated with more than one class of data? If so, is there a reason for this redundancy?

- Are any data items repeated within the same class of data? If so, can the data base be normalized?

Examination of the data items associated with "projects" and those associated with "employs" shows that both classes contain an employee number, an employee name, and a department number. Note that in the case of the "projects" data, the employee information is that of the project manager.

Redundant data such as this can use extra storage and complicate updates. If, for example, a project manager's department number changes, you must correct both the "projects" and "employs" data, leading to extra work and possible mistakes.

A simple solution is to remove the employee information, except for the project manager's identification number, from the "projects" data. You should leave the project manager's employee number with the projects data to provide a link to the "employs" data. With this link, detailed information about the project can be maintained separately from that of the employees, yet their association can be preserved.

Notice also that the same data item (an employee name) appears both in the "events" data and the "employs" data. In this case, the employee information is not only redundant; it is also confusing. If more than one employee has the same name, you cannot clearly identify the employees assigned to an event. To eliminate the confusion, replace the employee name in the "events" data with an employee number. Although this solution eliminates the confusing data, it does not remove repeating information from the "events" data. Since more than one employee can be assigned to an event, more than one "emp-no" could be associated with each event. The next step in the development process provides a solution to the problem of repeating data.

Identifying the Relationships Between Classes of Data

Identifying the relationship between classes of data helps to isolate and solve problems of association between classes. To identify the relationships, draw a diagram that shows each class of data along with a brief explanation of connections between them. Figure 12-4 shows such a diagram.

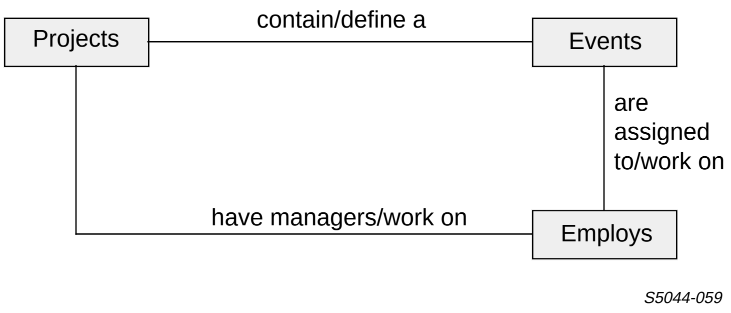


Figure 12-4. Relationships Between Classes of Data

After drawing the diagram, determine how many instances of one class relate to instances of another. To illustrate, consider the relationship between the "projects" data and the "employs" data. Each project can have only one project manager. Therefore, the relationship between these classes of data is called one-to-one (abbreviated to 1:1).

The relationship between the "projects" data and the "events" data is more complex. A project can have many events while an event can have only one project. This type of relationship is called a one-to-many relationship (in shorthand, 1:M).

The relationship between the "events" data and the "employs" data is even more complex. An event can have many employees and an employee can have many events. Thus, the relationship between these two classes of data is many-to-many (or M:M).

Figure 12-5 shows a diagram of these relationships.

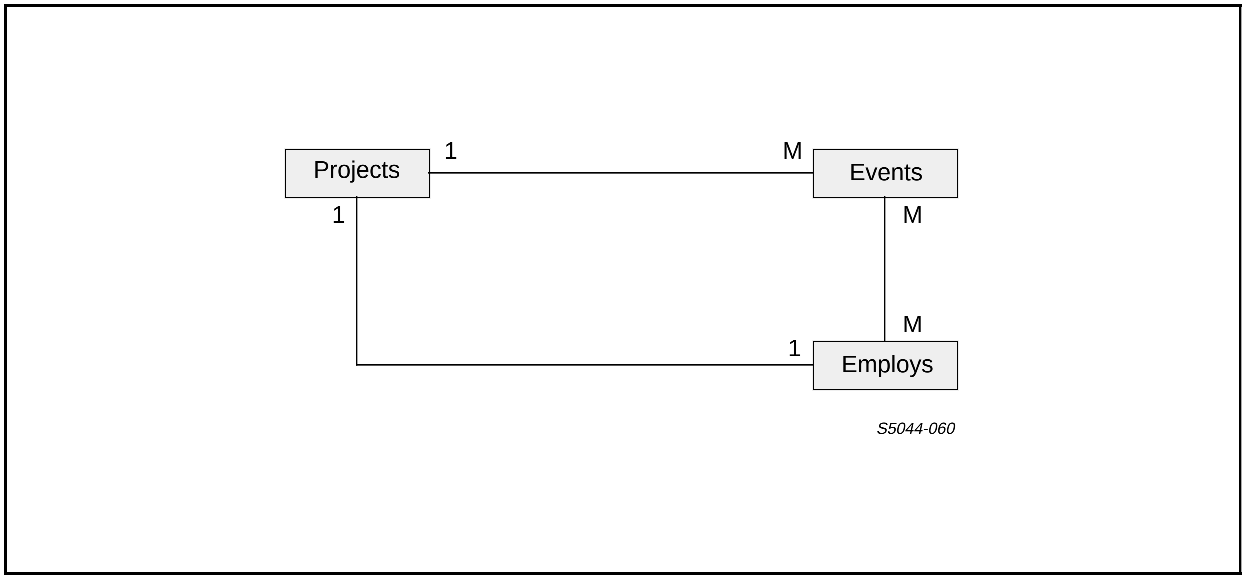


Figure 12-5. Relationships Between Instances of Classes of Data

To summarize, the relationships shown in Figure 12-5 are:

One project has one manager	1:1	
One project has many events	1:M	Each event has one project
One event can have many employees	M:M	One employee can have many events

Depending upon the type of relationship, the following rules may apply:

- With the exception of items involved in more than one relationship, data items that belong to classes with a one-to-one relationship can often be merged into a single class. Since the "employs" data is also related to the "events" class, the "projects" and "employs" data should not be merged.

- In one-to-many relationships, data items that apply to both classes can be included in the "one-occurrence" class to save space in the files. The data items that identify unique entries for the "one-occurrence" class can be included in both--to provide a link between the classes.

  To associate the "projects" and "events" data, include an item within the "events" class that uniquely identifies a project, such as "proj-code."

- All many-to-many relationships should be treated as a separate class of data.

  This rule offers a solution to the repeating data problem. If you create a separate class of data that contains an identifying item from "events" and an identifying item from "employs," you can remove the repeating data (the employee identification number) from "events."

Figure 12-6 shows a diagram that includes the new class of data.

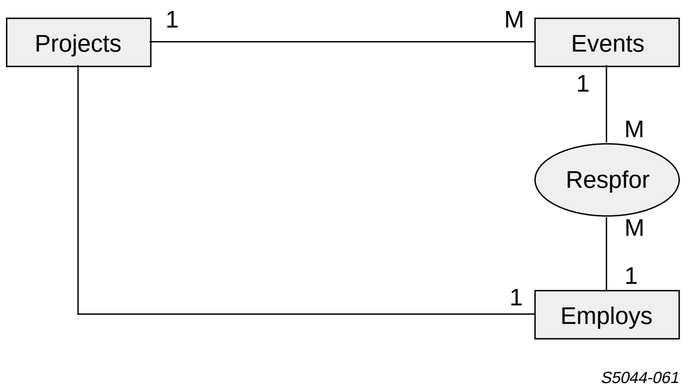


Figure 12-6. Relationships With New Class of Data

At this point, you have identified the files that you need for the project-tracking system. Each data class ("projects," "events," "employs," and now "respfor") corresponds to a file.

Listing the Fields in Each File

Having identified the files, you can list the fields in each. Figure 12-7 lists the fields in the files used by the project-tracking system.

File	Fields
Projects	proj-code, proj-desc, proj-status, proj-start-date, proj-end-date, proj-mgr
Events	proj-code, event-num, event-desc, predict-start-date, predict-end-date, revised-start-date, revised-end-date, change-info, update-emp
Respfor	proj-code, event-num, emp-no
Employs	emp-no, emp-name, dept-name

Figure 12-7. List of Fields in Each File

Notice that fields such as "proj-code," "event-num," and "emp-no" appear in a number of files. This duplication of fields is essential to link the files for a multifile application.

Identifying Key Fields

A primary key field is any field with a value that uniquely identifies a record. In the projects file, for example, there can be only one record for each "proj-code." Thus, "proj-code" can be the primary key for the "projects" file. Similarly, an "emp-no" uniquely identifies a record in the "employs" file.

In the "events" and "respfor" files, however, neither a "proj-code" nor an "event-num" alone is sufficient to identify a record uniquely. Each record in those files can be uniquely identified only by a combination of both "proj-code" and "event-num." Therefore, the primary key for these files must be a composite key group consisting of both "proj-code" and "event-num."

In addition to a primary key for each file, you may want to identify one or more alternate keys. The value of an alternate key field need not uniquely identify a record, but instead, provides an access path sorted in an alternate order. For example, while the "proj-desc" field of the "projects" file may have duplicate values, you can use this field as an alternate key field to display all projects with the same description.

When a key from one file appears as a field in a related file, you should make that field an alternate key in the related file as well. This provides you with the means to link the files within a multifile application.

Figure 12-8 shows a chart that illustrates the possible relationships between the files and keys used by the project-tracking system.

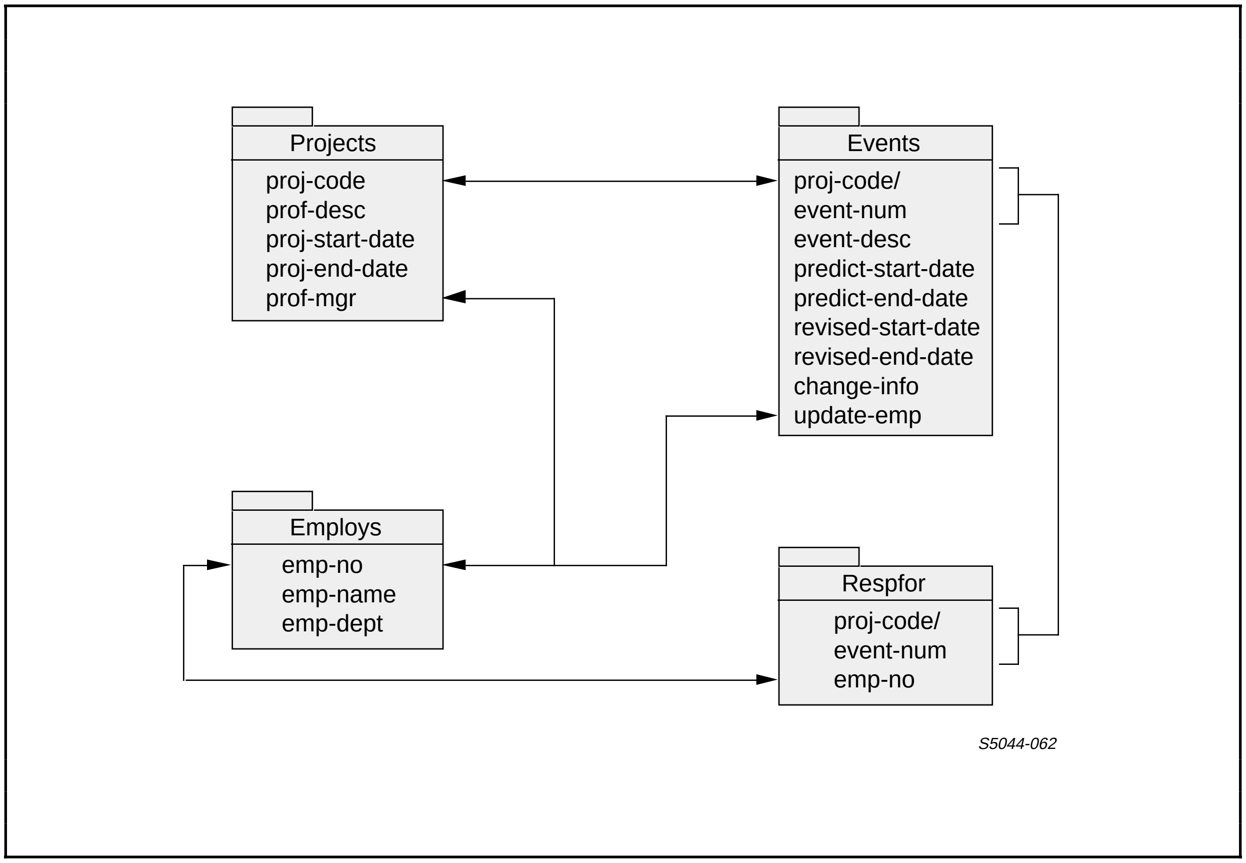


Figure 12-8. Possible Relationships Between Files and Keys

Checking for Normalization

A relational data base is usually normalized. In the simplest terms, this means that no file in the data base contains repeating data. As Figure 12-8 shows, no file used by the sample project-tracking system contains repeating groups. Therefore, these files are sufficiently normalized for use by an ENABLE application.

If you are developing your own application, refer to Section 3 for a discussion of the normalization process.

Creating a Dictionary That Describes the Files

Once you identify the files to be used by the system, you are ready to create a data dictionary that describes them. Figure 12-9 shows the DDL source code that creates the data dictionary for the project-tracking files.

```
RECORD projects.
FILE IS projects       KEY-SEQUENCED.
02 proj-code           PIC 9(6)  HEADING "Project Code".
02 proj-desc           PIC X(20) HEADING "Project Name".
02 proj-status         PIC X(5)  HEADING "Project Status".
02 proj-start-date               HEADING "Starting Date".
  04 start-day         PIC 99    HEADING "Day".
  04 start-mo          PIC 99    HEADING "Month".
  04 start-yr          PIC 99    HEADING "Year".
02 proj-end-date                 HEADING "Ending Date".
  04 end-day           PIC 99    HEADING "Day".
  04 end-mo            PIC 99    HEADING "Month".
  04 end-yr            PIC 99    HEADING "Year".
02 proj-mgr            PIC 9(6)  HEADING "Manager".
KEY 0 IS proj-code.
KEY "pd" IS proj-desc.
KEY "pm" IS proj-mgr.
END

RECORD employees.
FILE IS employs  KEY-SEQUENCED.
02 emp-no     PIC 9(6)  HEADING "Emp ID.".
02 emp-name   PIC X(30) HEADING "Employee Name".
02 emp-dept   PIC 9(4)  HEADING "Dept.".
KEY 0 IS emp-no.
KEY "en" IS emp-name.
KEY "ed" IS emp-dept.
END
```

Figure 12-9. DDL Source Code (Continued next page)

```
RECORD participants.
FILE IS respfor    KEY-SEQUENCED
02 event-key                      HEADING "Event ID".
   04 proj-code       PIC 9(6)   HEADING "Project No.".
   04 event-num       PIC 9(5)   HEADING "Event No.".
02 emp-no             PIC 9(6)
                      HEADING    "ID of Employee Assigned".
KEY 0 IS event-key.
KEY "en" IS emp-no.
END

RECORD events.
FILE IS events  KEY-SEQUENCED.
02 event-key                      HEADING "Event ID".
   04 proj-code       PIC 9(6)   HEADING "Project No.".
   04 event-num       PIC 9(5)   HEADING "Event No.".
02 event-desc         PIC X(20)  HEADING "Event".
02 predict-start-date HEADING    "Predicted Starting Date".
  04 start-day        PIC 99     HEADING "Day".
  04 start-mo         PIC 99     HEADING "Month".
  04 start-yr         PIC 99     HEADING "Year".
02 predict-end-date   HEADING    "Predicted Ending Date".
  04 end-day          PIC 99     HEADING "Day".
  04 end-mo           PIC 99     HEADING "Month".
  04 end-yr           PIC 99     HEADING "Year".
02 revised-start-date HEADING   "Revised Starting Date".
  04 start-day        PIC 99     HEADING "Day".
  04 start-mo         PIC 99     HEADING "Month".
  04 start-yr         PIC 99     HEADING "Year".
02 revised-end-date   HEADING   "Revised Ending Date".
  04 end-day          PIC 99     HEADING "Day".
  04 end-mo           PIC 99     HEADING "Month".
  04 end-yr           PIC 99     HEADING "Year".
02 change-info        PIC X(30)
                      HEADING    "Reason for Revising Date:".
02 update-emp         PIC 9(6)
                      HEADING "Update Employee ID".
KEY 0 IS event-key.
KEY "ue" IS update-emp.
KEY "ed" IS event-desc.
END
```

Figure 12-9. DDL Source Code (Continued)

Many of the fields in each record description have a DDL HEADING clause that you can use as a screen-field label when you generate an application.

DEFINING THE APPLICATIONS TO BE GENERATED

After you create the data dictionary, you can begin to specify the applications to be generated by performing the following steps:

1. Identify the purpose of each application.

2. For each application, list the files to be accessed, the fields that are to appear on the screen, and the operations that are to be permitted on each file.

3. Determine the format of the screen.

4. Identify any applications for which an automatic read operation would be appropriate.

5. Identify those applications, if any, that are to call others.

Identifying the Applications

By using the requirements that you determined earlier in the development process, you can identify the following applications to be generated:

1. "Project-entry"--an application to enter information about a project and the events associated with it

2. "Employee-assign"--an application used to enter data that associates an employee with a particular event

3. "Look-up"--an application that can be used to enter information about employees, including names and employee numbers

4. "Project-info"--an application that displays general information about a project and the events associated it

5. "Event-detail"--an application that displays detailed information about each event within a project

6. "Event-revised"--an application that allows an employee to revise information about a particular event

Identifying the Files to be Accessed by Each Application

Having identified the applications, make a rough sketch of the screens that are to be displayed by each, using a box to represent each file to be accessed by the application. In the sketch of the PROJECT-ENTRY screen, for example, you would include a box for the following files: "projects," "events," and "employs."

Next, determine the kinds of operations that the application will need to perform. Since a project manager using the "project-entry" application might need to delete, insert, or update information in the "projects" and "events" files, this application requires the ability to perform all of these operations. On the other hand, a project manager needs only to read information from the "employs" file. Therefore, the "project-entry" application requires only read access to this file. If you need to limit the kinds of operations that an application can perform on a particular file, include this information in your rough sketch of the application's terminal screen.

Figure 12-10 shows rough sketches of the screens for the project-tracking system.

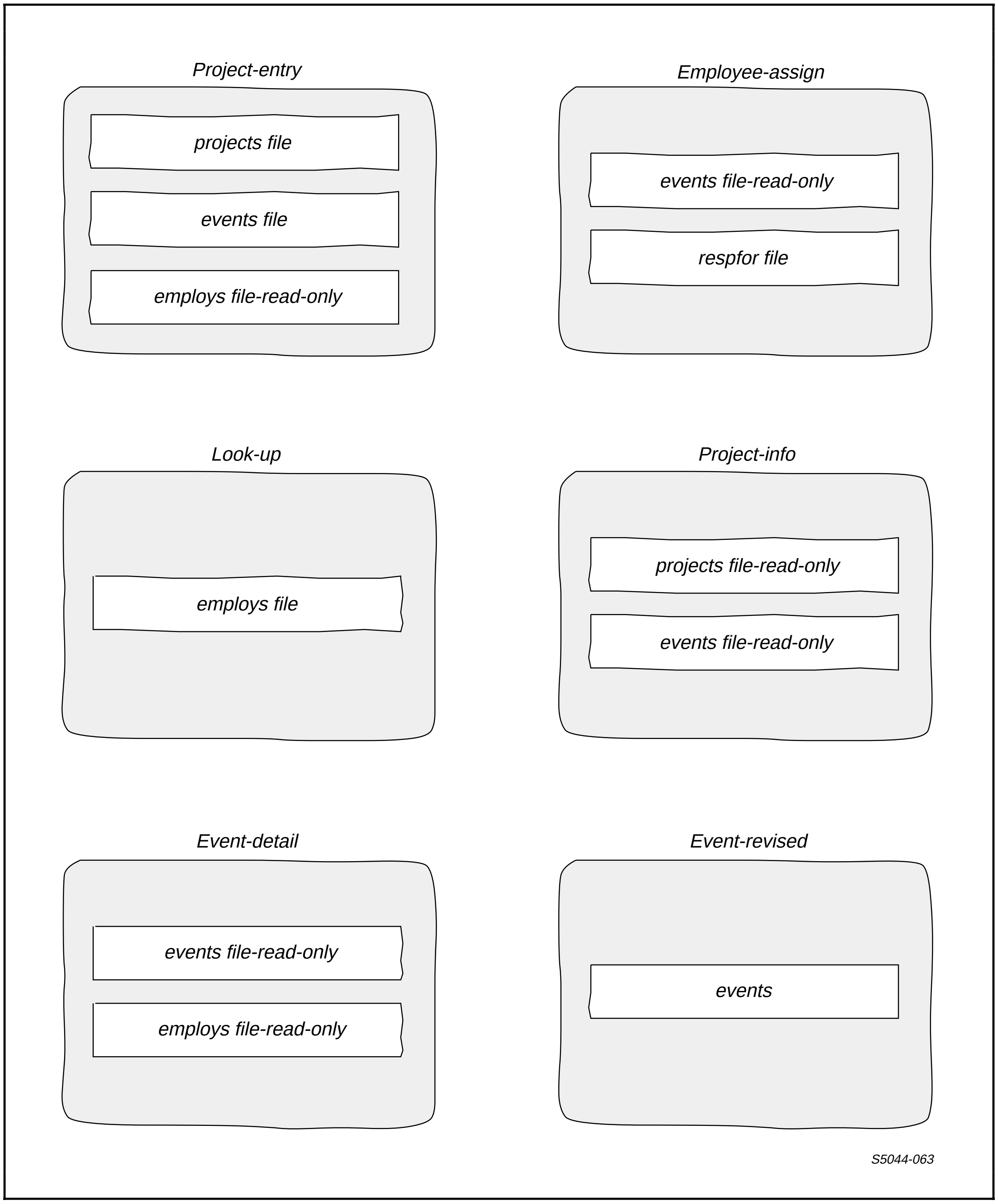


Figure 12-10. Rough Sketches of Screens in Project-Tracking System

When you define the organization of a tree, visualize the effect this has on the layout of the screen.

Since the purpose of the "project-entry" application is to enter and maintain information about projects and events, you might assume that a box representing the "projects" file should be the first to appear in the tree structure. The intended users of this application, however, are project managers who will have to enter their employee numbers in the "projects" file. You can make this task easier by making the box that represents the "employs" file the first box in the tree structure. If you link this box to the "projects" box using "emp-no" and "proj-mgr" (the field for the project manager's employee number) as the join fields, the application will automatically insert the project manager's employee number in the "projects" file.

Determining the Format of the Screens

After you define the tree structure for each application, you can define the detailed format of each screen, using the following guidelines:

- The first field that appears on the screen should be a key field.

  This guideline is particularly appropriate for applications whose primary function is to display information. Consider the "project-info" application. Its purpose is to display information about a project and the events associated with that project. A user of this application can expect to enter a value in the first screen field, press a function key, and see the desired information.

- For a multifile application, arrange the screen so that all fields appear within a box before any join fields.

  Not only is the screen more attractive, the applications are easier to use.

- Consider excluding fields from the screen if a user of the application does not need them.

  When a project manager uses the "project-entry" application to enter information about a project and its events, there is no need to enter information in the "revised-start-date," "revised-end-date," "change-info," or "update-emp" fields.

- If necessary, consider using more than one box to represent different fields from a file.

  You can improve the appearance of the screen by using more than one box to represent a file. You could, for example, use two boxes to display fields from the "events" file for the "event-detail" application.

  By using two boxes to represent the same file, you can restrict access to certain fields within the file while allowing full access to other fields within the same file. Consider the "event-revised" application, which uses information from a single file: the "events" file. While an employee who uses this application might want to see the contents of the "predict-start-date" field and the "predict-end-date" fields, you probably want to protect the contents of these fields from changes by the employee.

  If you restrict the "event-revised" application to read-only access to the events file, you protect the "predict-start-date" and "predict-end-date" fields, but you also preclude changes to the "revised-start-date" or "revised-end-date" fields. If you generate the application so that more than one box represents the "events" file and link these boxes on the primary key, the application will display a portion of the same record in each box. For example, a box with read-only access can display the "predict-start-date" and "predict-end-date" fields while a box with read and update access can display the "revised-start-date" and "revised-end-date" fields. Refer the description of generating the "event-revised" application for more information about this process.

- Provide descriptive screen titles.

  Each of the applications in the project-tracking system displays a screen title that describes its purpose. For example, the screen title of the look-up application is "Employee ID Display and Entry Screen."

- If appropriate, display any unusual user instructions on the screen.

  Within the project-tracking system, every calling application displays a message that tells the user how to request the called application; every called application displays a message that tells the user how to return to the calling application.

- If you want to display more than one record within a box, consider using a tabular format.

  Since a project can have more than one event, the "project-info" application should display more than one "events" record. A tabular format can display summary several records attractively.

When you have selected a screen layout for each application, make a detailed sketch of each proposed screen (see Figure 12-11) and show the sketch to the people who will use it. Wherever possible, incorporate any user suggestions into your screen design.

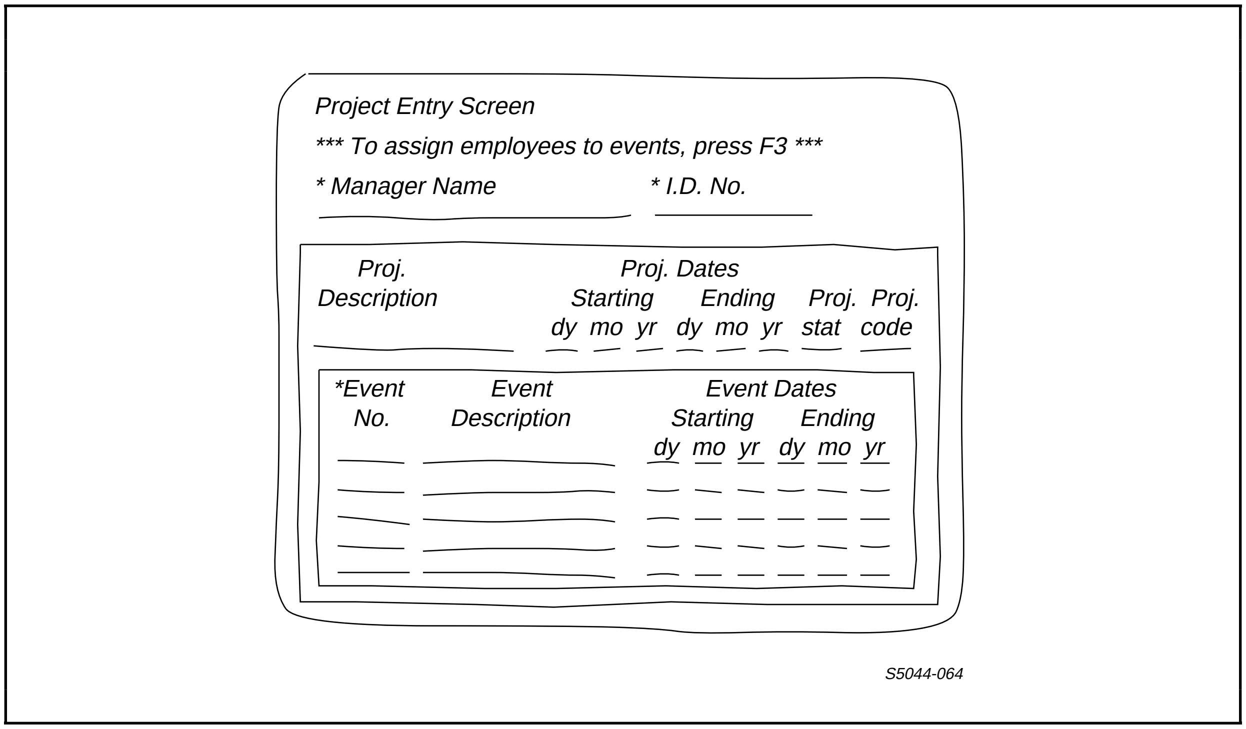


Figure 12-11. Detailed Sketch of the Proposed Project-Entry Screen

Identifying Candidate Applications for Automatic READ Operations

When the primary purpose of an application is to display information, most users expect to enter a value in a key field, press a function key, and view a screen full of information. To provide this expected behavior, you can supply ON as a value for the FILL attribute for every box (except the box at the first level of the tree structure) used by the application.

Figure 12-12 shows the applications (and the boxes within these applications) that should have FILL ON.

Application	Box That Represents:
PROJECT-INFO	The events file
EVENT-DETAIL	The events file The employs file
EVENT-REVISED	The copy of the events file for the: proj-code event-num predict-start-date predict-end-date

Figure 12-12. Applications With Automatic READ Operations

Identifying Calling Applications

The last task to perform before generating the applications is to identify those that need to call other applications, by asking the following questions about each:

- Is the purpose of this application to display general information about a particular subject? If so, does another application display detailed information about the same subject?

  The purpose of the "project-info" application is to display general information about a project. If a user wants detailed information about an event associated with that project, he or she must use the "event-detail" application. For this reason, a user of the "project-info" application should be able to call the "event-detail" application.

- Is one purpose of this application to enter data? If so, will some users of the application need information from another application to complete an entry in this one?

  The purpose of the "employee-assign" application is to enter data that assigns an employee to a particular event within a project. When using this application, a project manager enters an employee identification number to assign an employee to an event. If the project manager does not have a current list of employee numbers, he or she must use the "look-up" application to obtain the correct number. For this reason, a user of the "employee-assign" application should be able to call the "look-up" application.

- Is there a logical order in which the applications should be used?

  The purpose of the "project-entry" application is to enter information about a project and the events associated with it. After entering this information, a project manager can assign employees to each event. A project manager cannot assign an employee to an event until that event exists. For this reason, a user should be able to call the "employee-assign" application only from the "project-entry" application.

GENERATING THE APPLICATIONS

The next step in the development process is to generate the applications. The following paragraphs briefly discuss the ENABLE commands used to generate each of the following applications in the project-tracking system.

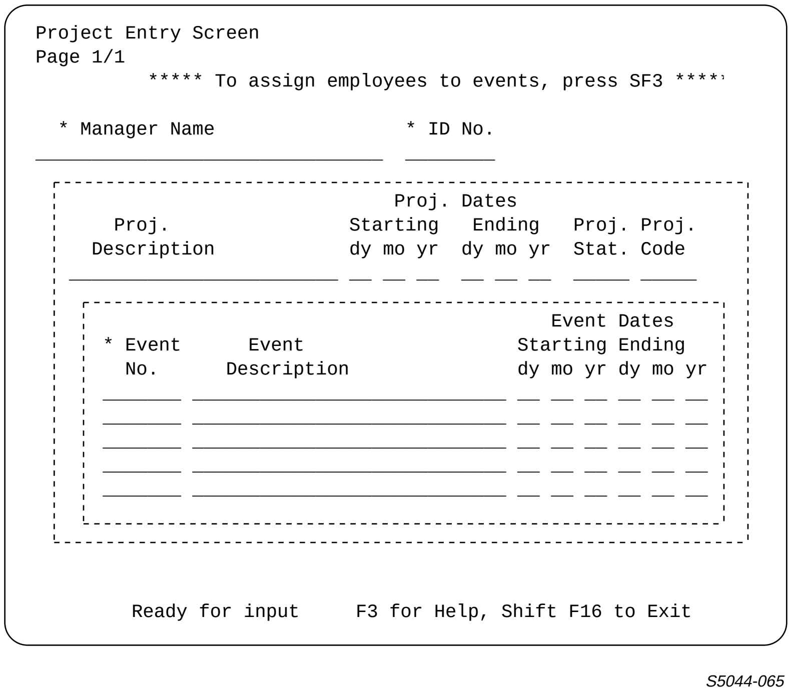
- Project-entry
- Employee-assign
- Look-up
- Project-info
- Event-detail
- Event-revised

Project-Entry Application

The purpose of the "project-entry" application is to allow a project manager to:

- Enter information about a new project and the events associated with it
- Update information about existing projects

This application accesses three files: "employs," "projects," and "events." Figure 12-13 shows the screen this application displays.



```
Project Entry Screen
Page 1/1
          ***** To assign employees to events, press SF3 *****

  * Manager Name                * ID No.
_______________________________  ________

                              Proj. Dates
     Proj.                Starting   Ending   Proj. Proj.
   Description            dy mo yr  dy mo yr  Stat. Code
________________________  __ __ __  __ __ __  _____ _____

                                            Event Dates
     * Event      Event                  Starting Ending
       No.     Description               dy mo yr dy mo yr
     _______ ____________________________ __ __ __ __ __ __
     _______ ____________________________ __ __ __ __ __ __
     _______ ____________________________ __ __ __ __ __ __
     _______ ____________________________ __ __ __ __ __ __
     _______ ____________________________ __ __ __ __ __ __

           Ready for input     F3 for Help, Shift F16 to Exit
```

S5044-065

Figure 12-13. Screen Displayed by Project-Entry Application

Figure 12-14 shows the ENABLE commands used to generate the "project-entry" application. The numbers to the right identify commands of particular interest that:

1. Establish a tabular format for the boxes used by the application

2. Reorder the screen fields displayed within each box (Note the fields for the "manager" box are reordered so that "emp-name," an alternate key field, appears first.)

3. Provide usage instructions

4. Restrict the application to read-only operations on records for the "manager" box, and allow delete, insert, read, and update operations on the records for the "projects" and "events" boxes

5. Indicate that the application can display up to five records in the "events" box

6. Request SCREEN COBOL source code that can be modified to provide the application with the ability to call "employee-assign"

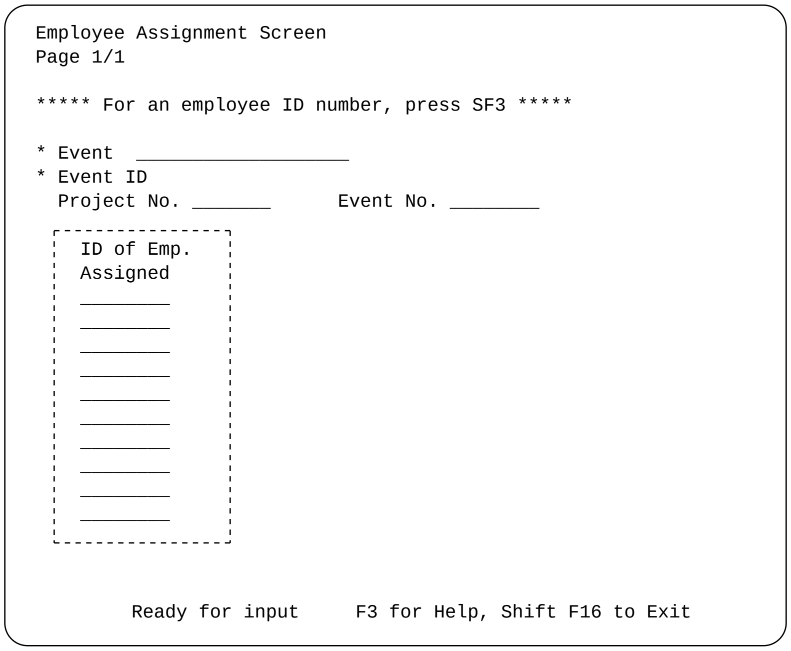
```
ASSUME BOX
SET SCREENFORMAT COMPRESSED  ①
SET HEADINGS NULL
SET RECORD employees
SET DELETE OFF, UPDATE OFF, INSERT OFF ④
SET INCLUDE (emp-name, emp-no) ②
SET BOXTITLE 1 "           ***** To assign employees to events,&
 press SF3 *****"                                                ③
SET BOXTITLE 3 "+   Manager Name               * ID No."
ADD manager
RESET DELETE, UPDATE, INSERT
SET RECORD projects
SET INCLUDE (proj-mgr, proj-desc, proj-start-date,
proj-end-date, proj-status, proj-code)
SET BOXTITLE 1 "                       Proj. Dates"
SET BOXTITLE 2 "      Proj.          Starting  Ending  Proj.  Proj."
SET BOXTITLE 3 "    Description      dy mo yr dy mo yr Stat.  Code"
ADD projects
SET SIZE 5 ⑤
SET RECORD events
SET INCLUDE (event-key, event-desc, predict-start-date,
predict-end-date)
SET BOXTITLE 1 "                              Event Dates "
SET BOXTITLE 2 "* Event       Events       Starting  Ending"
SET BOXTITLE 3 "  No.       Description    dy mo yr dy mo yr"
ADD events

ASSUME APPL
SET TREE (01 manager
  02 projects LINK emp-no TO OPTIONAL proj-mgr
     03 events LINK projects TO OPTIONAL events VIA proj-code)
SET TITLE "Project Entry Screen"
SET PATHCOMFILE prfile1 !
SET SCOBOLSOURCE appllsrc  ! ⑥
ADD project-entry
GENERATE project-entry
```

Figure 12-14. ENABLE Commands for the Project-Entry Application

Employee-Assign Application

The purpose of the "employee-assign" application is to allow a project manager to assign employees to particular events within a project. Since an event must exist before the project manager can use this application, he or she must call this application from the "project-entry" application instead of from the menu screen.

The "employee-assign" application accesses two files: "events" and "respfor." Figure 12-15 shows the screen displayed by this application.



```
Employee Assignment Screen
Page 1/1

***** For an employee ID number, press SF3 *****

* Event  ___________________
* Event ID
  Project No. _______      Event No. ________

    ID of Emp.
    Assigned
    ________
    ________
    ________
    ________
    ________
    ________
    ________
    ________
    ________
    ________

           Ready for input     F3 for Help, Shift F16 to Exit
```

S5044-066

Figure 12-15. Screen Displayed by Employee-Assign Application

Figure 12-16 shows the ENABLE commands, used to generate the "employee-assign" application, that:

1. Establish a compressed layout for the boxes used by the application

2. Reorder the fields from the "events" file and identify the fields that are to appear within "events-box"

3. Provide user information that appears within "events-box"

4. Reset the box attributes

5. Establish a tabular format for "partic-box" (The tabular format is established by the combined values of the SCREENFORMAT, SIZE, and HEADINGS attributes.)

6. Request the SCREEN COBOL source code that can be modified to provide the application with the ability to call "look-up"

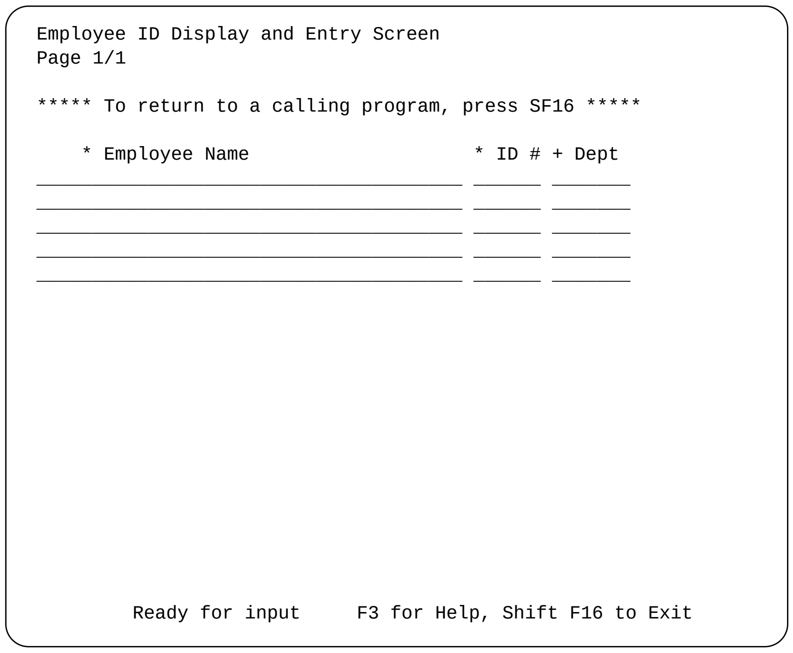
```
ASSUME BOX
SET RECORD events               ┐
SET HEADINGS DDLHEADINGS        ├─ 1
SET SCREENFORMAT COMPRESSED     ┘
SET DELETE OFF, INSERT OFF, UPDATE OFF
SET INCLUDE (event-desc, event-key)  2
SET BOXTITLE 2 " ***** For an employee ID number, press SF3 *****"  3
SET BOXTITLE 3 " "
ADD events-box
RESET BOX *   4
SET RECORD participants
SET HEADINGS NULL   5
SET INCLUDE (event-key, emp-no)
SET SIZE 10
SET BOXTITLE 1 "ID of Emp."
SET BOXTITLE 2 "Assigned"
ADD partic-box

ASSUME APPL
SET PATHCOMFILE prfile2 !
SET SCOBOLSOURCE exam2src !   6
SET TREE (01 events-box
  03 partic-box LINK events-box TO OPTIONAL partic-box VIA event-key)
SET TITLE "Employee Assignment Screen"
ADD employee-assign
GENERATE APPL employee-assign
```

Figure 12-16. ENABLE Commands to Generate the Employee-Assign Application

Look-Up Application

The purpose of the "look-up" application is to allow a user to display and update employee information. A user can call this application from either the "employee-assign" application or the menu screen.

The "look-up" application accesses the "employs" file. Figure 12-17 shows the screen it displays.



```
Employee ID Display and Entry Screen
Page 1/1

***** To return to a calling program, press SF16 *****

    * Employee Name                      * ID # + Dept
______________________________________ ______ _______
______________________________________ ______ _______
______________________________________ ______ _______
______________________________________ ______ _______
______________________________________ ______ _______

           Ready for input     F3 for Help, Shift F16 to Exit
```

Figure 12-17. Screen Displayed by the Look-Up Application

Figure 12-18 shows the ENABLE commands, used to generate the "look-up" application, that:

1. Provide a tabular screen format for the employee records

2. Provide usage information to appear within the "employees" box

3. Provide screen labels for the employee records

4. Request a SCREEN COBOL compilation listing that will be used when the application is integrated into a single PATHWAY system

```
ASSUME BOX
SET RECORD employees
SET HEADINGS NULL                 ┐
SET SCREENFORMAT COMPRESSED       ├─ ①
SET SIZE 5                        ┘
SET BOXTITLE 1 " ***** To return to a calling program, press SF16 *****"  ②
SET BOXTITLE 3 "     + Employee Name           * ID # + Dept."  ③
SET INCLUDE (emp-name, emp-no, emp-dept)
ADD employees
ASSUME APPL
SET TITLE "Employee ID Display and Entry Screen"
SET TREE (01 employees)
SET SCOBOLLIST exam31st !   ④
SET PATHCOMFILE prfile4 !
ADD look-up
GENERATE APPL look-up
```

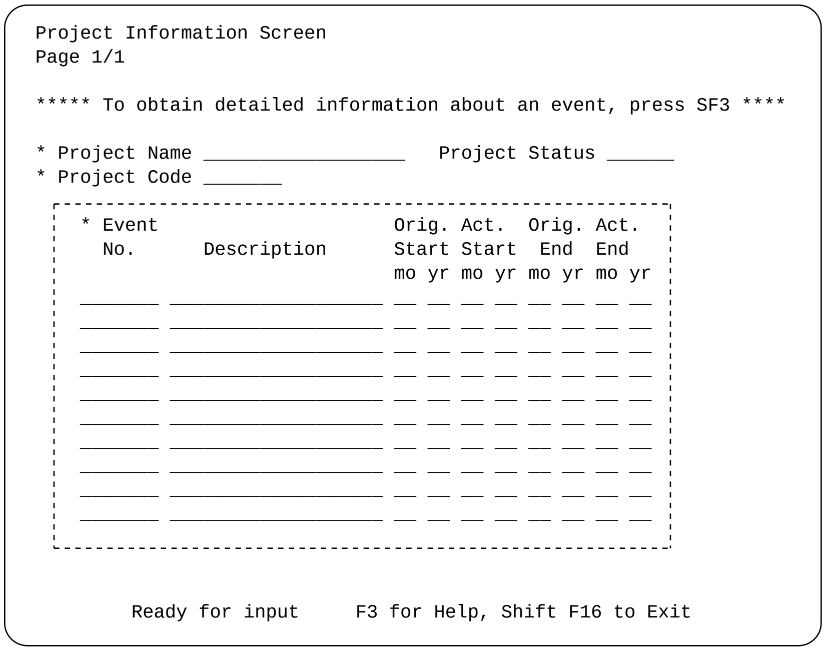
Figure 12-18. ENABLE Commands Used to Generate the Look-Up Application

Project-Info Application

The purpose of the "project-info" application is to display general information about a project and the events related to it. To use this application, enter a project name and press the appropriate read-operation key. The application returns information about the project and up to ten related events. If detailed information about an event is needed, the user can press a function key and call the "event-detail" application.

Sample Project-Tracking System
Project-Info Application

The "project-info" application accesses two data base files: "projects" and "events." Figure 12-19 shows the screen it displays.



```
Project Information Screen
Page 1/1

***** To obtain detailed information about an event, press SF3 ****

* Project Name __________________   Project Status ______
* Project Code _______

  * Event                           Orig. Act.  Orig. Act.
    No.      Description            Start Start  End  End
                                    mo yr mo yr mo yr mo yr
  _______ ___________________ __ __ __ __ __ __ __ __
  _______ ___________________ __ __ __ __ __ __ __ __
  _______ ___________________ __ __ __ __ __ __ __ __
  _______ ___________________ __ __ __ __ __ __ __ __
  _______ ___________________ __ __ __ __ __ __ __ __
  _______ ___________________ __ __ __ __ __ __ __ __
  _______ ___________________ __ __ __ __ __ __ __ __
  _______ ___________________ __ __ __ __ __ __ __ __
  _______ ___________________ __ __ __ __ __ __ __ __
  _______ ___________________ __ __ __ __ __ __ __ __

        Ready for input     F3 for Help, Shift F16 to Exit
```

S5044-068

Figure 12-19. Screen Displayed by the Project-Info Application

Figure 12-20 shows the ENABLE commands, used to generate the "project-info" application, that:

1. Restrict the application to read-only access to both the "projects" and "events" files

2. Request a compressed screen format

3. Reorder the fields for "project-box" so that the join field ("proj-code") appears last

4. In combination with the setting of SCREENFORMAT, these commands provide a tabular format for "events-box"

5. Indicate that the application is to perform an automatic read operation on "events-box" whenever the contents of "proj-box" changes

6. Request the SCREEN COBOL source code for this application. You must modify this source code to call the "event-detail" application

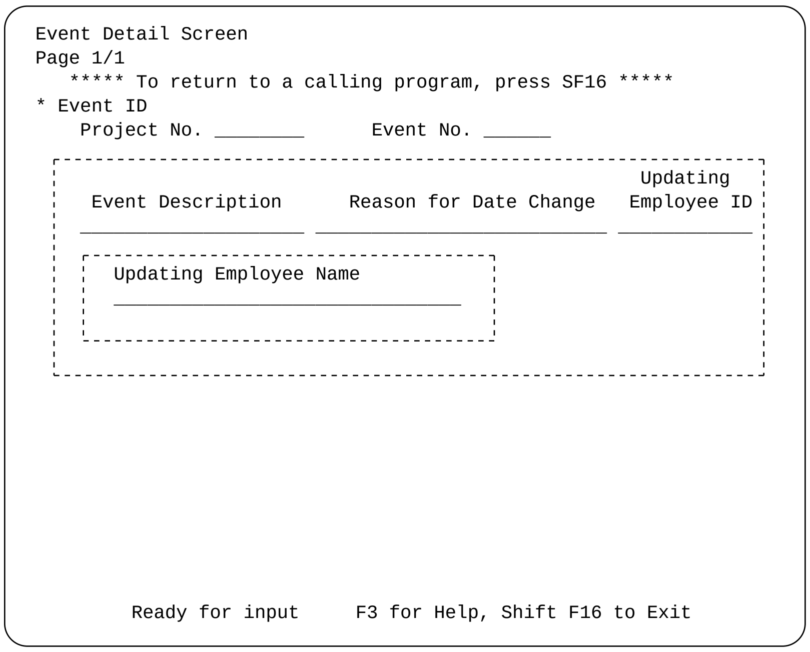
```
ASSUME BOX
SET DELETE OFF, INSERT OFF, UPDATE OFF  ①
SET SCREENFORMAT COMPRESSED  ②
SET RECORD project
SET HEADINGS DDLHEADINGS
SET INCLUDE (proj-desc,proj-status,proj-code)  ③
SET BOXTITLE 2 " ***** To obtain detailed information about an event, press
 SF3 *****"
SET BOXTITLE 3 " "
ADD proj-box
SET RECORD events  ┐
SET SIZE 10        ├─ ④
SET HEADINGS NULL  ┘
SET BOXTITLE 1"* Event                    Orig.  Act. Orig. Act."
SET BOXTITLE 2"  No.     Description      Start Start  End  End "
SET BOXTITLE 3"                           mo yr mo yr mo yr mo yr"
SET INCLUDE (event-key, event-desc,predict-start-date.start-mo,
predict-start-date.start-yr, revised-start-date.start-mo,
revised-start-date.start-yr, predict-end-date.start-mo,
predict-end-date.start-yr, revised-end-date.start-mo,
revised-end-date.start-yr)
SET FILL ON   ⑤
ADD events-box

ASSUME APPL
SET SCOBOLSOURCE exam4src  !  ⑥
SET PATHCOMFILE prfile4 !
SET TITLE "Project Information Screen"
SET TREE (01 proj-box
  02 events-box LINK project-box TO OPTIONAL events-box VIA proj-code)
ADD project-info
GENERATE project-info
```

Figure 12-20. ENABLE Commands to Generate the Project-Info Application

Event-Detail Application

The purpose of the "event-detail" application is to display detailed information about a particular event within a project. To use this application, a person enters a project number and an event number. When the user presses an appropriate read-operation key, the application returns a description of the event, the reason that the event dates were revised, and information that identifies the employee who revised the dates.

The "event-detail" application accesses two files: "events" and "employs." Figure 12-21 shows the screen it displays.



```
Event Detail Screen
Page 1/1
   ***** To return to a calling program, press SF16 *****
* Event ID
    Project No. ________      Event No. ______

                                                  Updating
  Event Description      Reason for Date Change  Employee ID
  ____________________  __________________________ ____________

   Updating Employee Name
   _______________________________

           Ready for input     F3 for Help, Shift F16 to Exit
```

Figure 12-21. Screen Displayed by the Event-Detail Application

Figure 12-22 shows the ENABLE commands used to generate the "events-detail" application. The numbers that appear to the right of this figure refer to commands of particular interest that:

1. Limit the application to read-only access to both the "events" and "employs" files

2. Supply "events" as the value of the RECORD attribute (Note that this value of the RECORD attribute applies to both the "event-x" box and the "events-a" box. Although both boxes represent the events file, each box displays different fields from the same record in this file.)

3. Indicate that the application is to perform an automatic read operation for the "events-a" and "employ-assign" boxes whenever the contents of the "events-x" box changes

4. Request a SCREEN COBOL compilation listing for this application. You will need information from this listing when you integrate the application into the project-tracking system

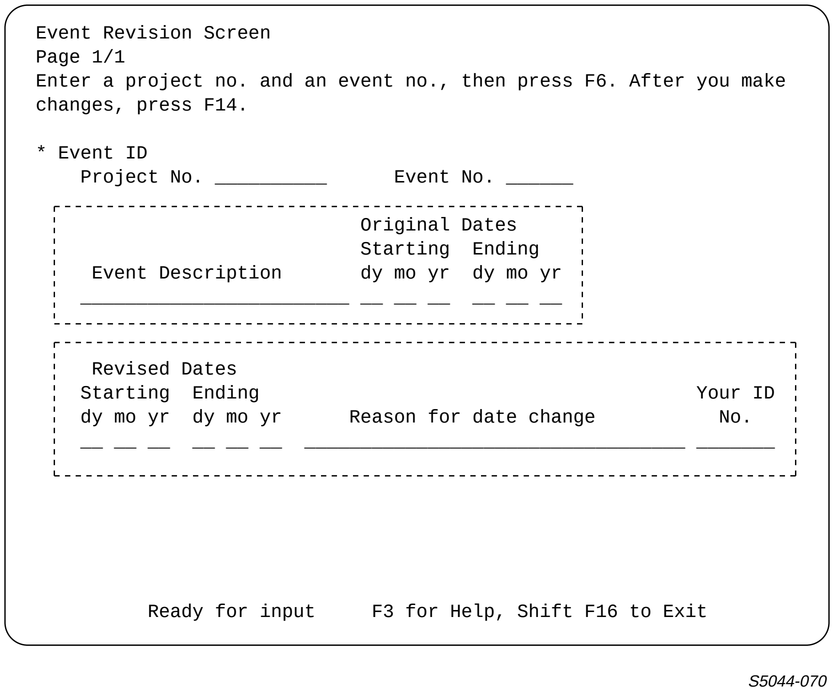
```
ASSUME BOX
SET DELETE OFF, INSERT OFF, UPDATE OFF  ①
SET SCREENFORMAT COMPRESSED
SET RECORD events  ②
SET HEADINGS DDLHEADINGS
SET BOXTITLE 1 " ***** To return to a calling program, press SF16 *****"
SET INCLUDE (event-key)
ADD event-x
SET FILL ON  ③
SET INCLUDE (event-key, event-desc, change-info, update-emp)
SET HEADINGS NULL
SET BOXTITLE 1 "                                                     Updating"
SET BOXTITLE 2 "  Event Description     Reason for Date Change      Employee ID"
ADD events-a
RESET BOXTITLE 2
SET RECORD employees
SET INCLUDE (emp-no, emp-name)
SET BOXTITLE 1 "Updating Employee Name"
ADD employ-assign

ASSUME APPL
SET PATHCOMFILE prfile5 !
SET SCOBOLLIST exam51st !  ④
SET TREE (01 event-x
  02 events-a LINK event-x TO OPTIONAL events-a VIA event-key
  03 employ-assign LINK update-emp TO OPTIONAL emp-no)
SET TITLE "Event Detail Screen"
ADD event-detail
GENERATE event-detail
```

Figure 12-22. ENABLE Commands to Generate the Event-Detail Application

Event-Revised Application

The purpose of the "event-revised" application is to provide employees with the ability to revise the starting and ending dates for a particular event within a project. To find the correct event, the employee enters a project number and an event number. The application then returns the event description and the predicted starting and ending dates. Although the employee cannot change the predicted starting and ending dates, he or she can enter revised dates and describe the reason for the date change.

The "event-revised" application uses three different boxes to represent the "events" file. When an employee uses the application, each box contains a different portion of the same events record. Figure 12-23 shows the screen displayed by this application.



```
Event Revision Screen
Page 1/1
Enter a project no. and an event no., then press F6. After you make
changes, press F14.

* Event ID
    Project No. __________      Event No. ______

                             Original Dates
                             Starting  Ending
   Event Description         dy mo yr  dy mo yr
  ________________________ __ __ __  __ __ __

   Revised Dates
  Starting  Ending                                    Your ID
  dy mo yr  dy mo yr      Reason for date change        No.
  __ __ __  __ __ __  __________________________________ _______

          Ready for input     F3 for Help, Shift F16 to Exit
```

Figure 12-23. Screen Displayed by Event-Revised Application

Figure 12-24 shows the ENABLE commands used to generate the "event-revised" application. The numbers that appear to the right of this figure refer to commands of particular interest that:

1. Identify "events" as the value of the RECORD attribute (Since this value is neither overridden or reset, ENABLE uses this value for the record description of all the boxes used by the application.)

2. Indicate that the application can only read records for the "my-events" and "events-3" boxes (These boxes display fields from an event record that should not be changed.)

3. Indicate that the application is automatically to read information for the "events-3" box whenever the contents of the "my-events" box changes

4. Reset the UPDATE attribute to allow a user to update the fields from an "events" record displayed within the "events-4" box

5. Request a SCREEN COBOL compilation listing for this application

```
ASSUME BOX  ①
SET RECORD events
SET SCREENFORMAT COMPRESSED
SET HEADINGS DDLHEADINGS
SET DELETE OFF, INSERT OFF, UPDATE OFF  ②
SET INCLUDE (event-key)
SET BOXTITLE 1 "Enter a project no. and an event no., then press F6.  After you make"
SET BOXTITLE 2 "changes, press F14. "
SET BOXTITLE 3 " "
ADD BOX my-event
SET FILL ON    ③
SET HEADINGS NULL
SET INCLUDE (event-key, event-desc, predict-start-date, predict-end-date)
SET BOXTITLE 1 "                     Original Dates  "
SET BOXTITLE 2 " Event Description   Starting Ending "
SET BOXTITLE 3 "                     dy mo yr dy mo yr"
ADD BOX events-3
RESET UPDATE    ④
SET INCLUDE (event-key, revised-start-date, revised-end-date, change-info,
update-emp)
SET BOXTITLE 1" Revised Dates "
SET BOXTITLE 2"Starting Ending                                  Your ID"
SET BOXTITLE 3"dy mo yr dy mo yr   Reason for Date Change         No.  "
ADD BOX events-4
SET APPL PATHCOMFILE prfile6 !    ⑤
SET APPL SCOBOLLIST exam61st !
SET APPL TREE (01 my-event
  03 events-3 LINK my-event TO OPTIONAL events-3 VIA event-key
  03 events-4 LINK my-event TO OPTIONAL events-4 VIA event-key)
SET APPL TITLE "Event Revision Screen"
ADD APPL event-revised
GENERATE APPL event-revised
```

Figure 12-24. ENABLE Commands to Generate the Event-Revised Application

MODIFYING THE SCREEN COBOL SOURCE CODE OF SELECTED APPLICATIONS

Three applications within the project-tracking system have the ability to call other applications. These calling applications are "project-entry," "employee-assign," and "project-info."

To allow these applications to call others, you must modify and compile their source code files, as described in Section 11. Briefly, you must change an entry in the T9155-CHAIN paragraph of the source code to include a call to the appropriate application.

Modifying the Source Code for the Project-Entry Application

The "project-entry" application needs the ability to call the "employee-assign" application. The T9155-CHAIN paragraph of the "project-entry" source code appears as follows:

```
   T9155-CHAIN.
     IF T9155-MANAGERBOX-CHOICE
       MOVE T9155-NOT-SUPPORTED-MESSAGE TO T9155-ERROR-MSG
     ELSE IF T9155-PROJECTBOX-CHOICE
       MOVE T9155-NOT-SUPPORTED-MESSAGE TO T9155-ERROR-MSG
     ELSE IF T9155-EVENTSBOX-CHOICE
       MOVE T9155-NOT-SUPPORTED-MESSAGE TO T9155-ERROR-MSG
     ELSE NEXT SENTENCE.
```

To allow a user to call the "employee-assign" application when the cursor is positioned within the "events" box of "project-entry," the source code must be modified to include the CALL statement as follows:

```
   T9155-CHAIN.
     IF T9155-MANAGERBOX-CHOICE
       MOVE T9155-CANT-CHAIN--MESSAGE TO T9155-ERROR-MSG
     ELSE IF T9155-PROJECTBOX-CHOICE
       MOVE T9155-CANT-CHAIN-MESSAGE TO T9155-ERROR-MSG
     ELSE IF T9155-EVENTSBOX-CHOICE
       CALL employee-assign
     ELSE NEXT SENTENCE.
```

Notice that a T9155-CANT-CHAIN-MESSAGE replaces the T9155-NOT-SUPPORTED-MESSAGE for both MANAGERBOX-CHOICE and PROJECTBOX-CHOICE. If the user must position the cursor with a particular box to call another application, be sure to exchange these messages. Refer to Section 11 for more information about the T9155-CANT-CHAIN-MESSAGE.

Modifying the Source Code for the Employee-Assign Application

The "employee-assign" application needs the ability to call the "look-up" application. The T9155-CHAIN paragraph for the "employee-assign" application appears as follows:

```
T9155-CHAIN.
  IF T9155-EVENTSBOX-CHOICE
    MOVE T9155-NOT-SUPPORTED-MESSAGE TO T9155-ERROR-MSG
  ELSE IF T9155-PARTICBOX-CHOICE
    MOVE T9155-NOT-SUPPORTED-MESSAGE TO T9155-ERROR-MSG
  ELSE NEXT SENTENCE.
```

To allow users of this application to call the "look-up" application, you must modify this source code to appear as follows:

```
T9155-CHAIN.
  IF T9155-EVENTSBOX-CHOICE
    CALL look-up
  ELSE IF T9155-PARTICBOX-CHOICE
    CALL look-up
  ELSE NEXT SENTENCE.
```

Modifying the Source Code for the Project-Info Application

The "project-info" application needs the ability to call the "event-detail" application. The T9155-CHAIN paragraph of the "project-info" source code appears as follows:

```
T9155-CHAIN.
  IF T9155-PROJECTBOX-CHOICE
    MOVE T9155-NOT-SUPPORTED-MESSAGE TO T9155-ERROR-MSG
  ELSE IF T9155-EVENTSBOX-CHOICE
    MOVE T9155-NOT-SUPPORTED-MESSAGE TO T9155-ERROR-MSG
  ELSE NEXT SENTENCE.
```

To allow users of this application to call the "event-detail" application, modify this source code to include the CALL statement as follows:

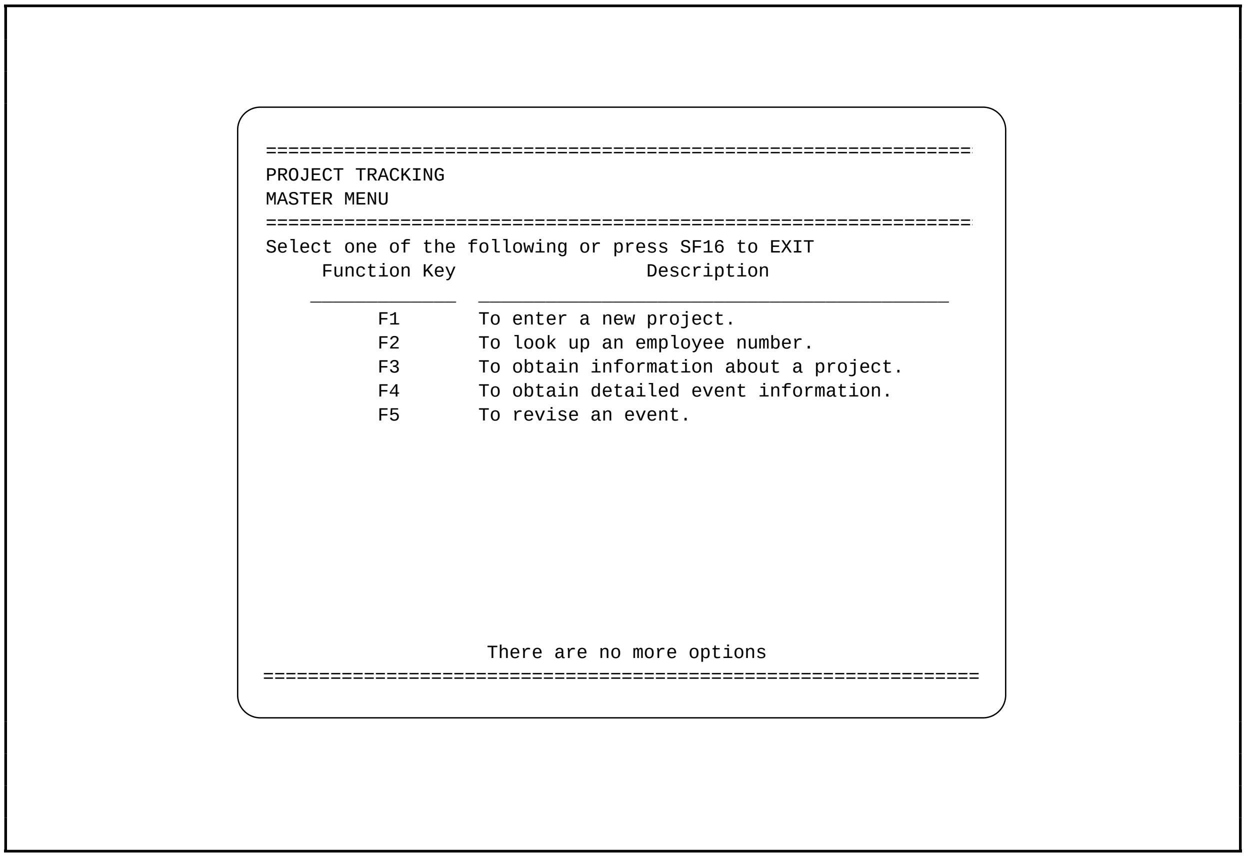
```
T9155-CHAIN.
  IF T9155-PROJECTBOX-CHOICE
    CALL event-detail
  ELSE IF T9155-EVENTSBOX-CHOICE
    CALL event-detail
  ELSE NEXT SENTENCE.
```

After making these modifications, you must compile the source code for each application using the SCOBOLX command discussed in Section 11. For example, if source code for the "project-entry" application is in a file named "exam1src," you compile this source code by entering:

```
   SCOBOLX/IN exam1src, OUT listex1, MEM 64, NOWAIT/ pobj
```

WRITING A MENU PROGRAM

After you have generated all the applications to be integrated into a single system, you can write a menu program that calls these applications. Figure 12-25 shows the screen displayed by a sample menu program for the project-tracking system.



```
==============================================================
PROJECT TRACKING
MASTER MENU
==============================================================
Select one of the following or press SF16 to EXIT
     Function Key               Description
    _____________  __________________________________________
          F1       To enter a new project.
          F2       To look up an employee number.
          F3       To obtain information about a project.
          F4       To obtain detailed event information.
          F5       To revise an event.




                   There are no more options
================================================================
```

Figure 12-25. Screen Displayed by Sample Menu Program

Figure 12-26 shows the SCREEN COBOL source code for this program. Within the source code, comments (indicated by an asterisk in column 1) provide additional information. You can modify this source code to develop a menu program for your own system of applications.

```
 IDENTIFICATION DIVISION.
  PROGRAM-ID.  MENU.
    AUTHOR.        Your Name
    DATE-WRITTEN.  06/29/84
    DATE-COMPILED.
    SECURITY.      comment.
 ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
   SOURCE-COMPUTER.  T16.
   OBJECT-COMPUTER.  T16,
*
* Modify the following for T16-651x or IBM-327x terminals.
*
    TERMINAL IS T16-6520.

   SPECIAL-NAMES.
     F1-KEY IS F1,
     F2-KEY IS F2,
     F3-KEY IS F3,
     F4-KEY IS F4,
     F5-KEY IS F5,
     F6-KEY IS F6,
     F7-KEY IS F7,
     F8-KEY IS F8,
     F9-KEY IS F9,
     F10-KEY IS F10,
     F11-KEY IS F11,
     F12-KEY IS F12,
     F13-KEY IS F13,
     F14-KEY IS F14,
     F15-KEY IS F15,
     F16-KEY IS F16,
     SF1-KEY IS SF1,
     SF2-KEY IS SF2,
     SF3-KEY IS SF3,
     SF4-KEY IS SF4,
     SF5-KEY IS SF5,
     SF6-KEY IS SF6,
```

Figure 12-26. SCREEN COBOL Source Code for Sample Menu Program
(Continued next page)

```
       SF7-KEY IS SF7,
       SF8-KEY IS SF8,
       SF9-KEY IS SF9,
       SF10-KEY IS SF10,
       SF11-KEY IS SF11,
       SF12-KEY IS SF12,
       SF13-KEY IS SF13,
       SF14-KEY IS SF14,
       SF15-KEY IS SF15,
       SF16-KEY IS SF16,
       NORMAL IS NORMAL,
       DIM IS DIM,
       REVERSE IS REVERSE,
       HIDDEM IS HIDDEN,
       BLINK IS BLINK,
       NOBLINK IS NOBLINK,
       NOREVERSE IS NOREVERSE,
       UNDERLINE IS UNDERLINE,
       ADVICE IS (DIM, REVERSE).

    INPUT-OUTPUT SECTION.
      SCREEN-CONTROL.
        ERROR-ENHANCEMENT IS REVERSE IN ALL.

    DATA DIVISION.
    WORKING-STORAGE SECTION.
    01 CALL-ERROR-MESSAGES.
 *        Termination status = 0001
      02  FILLER                    PIC X(32)
                 VALUE "Invalid Pseudocode detected     ".
 *        Termination status = 0002
      02  FILLER                    PIC X(32)
                 VALUE "Depending variable value to big ".
 *        Termination status = 0003
      02  FILLER                    PIC X(32)
                 VALUE "Invalid subscript value         ".
 *        Termination status = 0004
      02  FILLER                    PIC X(32)
                 VALUE "Screen Recovery -illegal instr. ".
 *        Termination status = 0005
      02  FILLER                    PIC X(32)
                 VALUE "Mismatch -  number of parameters".
 *        Termination status = 0006
```

Figure 12-26. SCREEN COBOL Source Code for Sample Menu Program (Continued)

```
       02  FILLER                    PIC X(32)
                  VALUE "Mismatch - size of parameters   ".
   *       Termination status = 0007
       02  FILLER                    PIC X(32)
                  VALUE "Screen operation without base   ".
   *       Termination status = 0008
       02  FILLER                    PIC X(32)
                  VALUE "Invalid data reference          ".
   *       Termination status = 0009
       02  FILLER                    PIC X(32)
                  VALUE "Screen illegal for term type    ".
   *       Termination status = 0010
       02  FILLER                    PIC X(32)
                  VALUE "Internal err in terminal format ".
   *       Termination status = 0011
       02  FILLER                    PIC X(32)
                 VALUE "Illegal terminal type specified ".
   *       Termination status = 0012
       02  FILLER                    PIC X(32)
                  VALUE "Screen referenced/not displayed ".
   *       Termination status = 0013
       02  FILLER                    PIC X(32)
                  VALUE "Overlay screen displayed/2 areas".
   *       Termination status = 0014
       02  FILLER                    PIC X(32)
                  VALUE "Illegal term IO protocol word   ".
   *       Termination status = 0015
       02  FILLER                    PIC X(32)
                  VALUE "Arithmetic Overflow             ".
   *       Termination status = 0016
       02  FILLER                    PIC X(32)
                  VALUE "Terminal stack space overflow   ".
   *       Termination status = 0017
       02  FILLER                    PIC X(32)
                  VALUE "Error during terminal open      ".
   *       Termination status = 0018
       02  FILLER                    PIC X(32)
                  VALUE "Error during terminal IO        ".
   *       Termination status = 0019
       02  FILLER                    PIC X(32)
                  VALUE "Wrong transfer count in term IO ".
   *       Termination status = 0020
       02  FILLER                    PIC X(32)
                  VALUE "Called program unit not found   ".
   *       Termination status = 0021
```

Figure 12-26. SCREEN COBOL Source Code for Sample Menu Program (Continued)

```
      02  FILLER                    PIC X(32)
                 VALUE "Transaction msg send failure    ".
  *       Termination status = 0022
      02  FILLER                    PIC X(32)
                 VALUE "Send:server class name invalid  ".
  *       Termination status = 0023
      02  FILLER                    PIC X(32)
                 VALUE "Pseudocode size too big         ".
  *       Termination status = 0024
      02  FILLER                    PIC X(32)
                 VALUE "TCLPROG directory entry is bad  ".
  *       Termination status = 0025
      02  FILLER                    PIC X(32)
                 VALUE "Term input data stream invalid  ".
  *       Termination status = 0026
      02  FILLER                    PIC X(32)
                 VALUE "Program unit has wrong term type".
  *       Termination status = 0027
      02  FILLER                    PIC X(32)
                 VALUE "Transaction mode violation      ".
  *       Termination status = 0028
      02  FILLER                    PIC X(32)
                 VALUE "Transaction IO error            ".
  *       Termination status = 0029
      02  FILLER                    PIC X(32)
                 VALUE "Trans. restart limit reached    ".
  *       Termination status = 0030
      02  FILLER                    PIC X(32)
                 VALUE "TMF not configured              ".
  *       Termination status = 0031
      02  FILLER                    PIC X(32)
                  VALUE "TMF not running                 ".
   01 ERROR-ARRAY REDEFINES CALL-ERROR-MESSAGES.
     02 CALL-ERROR-MESSAGE
                OCCURS 31 TIMES PIC X(32).

   01 ERROR-NOTE.
     02  FILLER                    PIC X(18)
                VALUE " Program called=> ".
     02  PROG-NAME          PIC X(15) VALUE "???????????????".
```

Figure 12-26. SCREEN COBOL Source Code for Sample Menu Program
(Continued)

```
 01 BELL.
   02  FILLER                       PIC 9 COMP VALUE 7.
*    ..................................................
 01 MAX-CHOICE                      PIC 99 VALUE 0.
 01 OPERATION-CHOICE                PIC 99 VALUE 1.
 01 TASK                            PIC X.
   88 TASK-DONE                    VALUE "Y".
   88 TASK-NOT-DONE                VALUE "N".
 01 NCOUNT                          PIC 99 VALUE 1.
 01 END-PROGRAM                     PIC X VALUE "N".
 01 PROGRAM-CALLED                  PIC X(15) VALUE SPACES.

************************************************************
*    PLACE THE NAMES OF THE CALLED PROGRAMS IN  THE VALUE
*    CLAUSES THAT FOLLOW.  PROG1 CORRESPONDS TO FUNCTION
*    KEY 1 AND DESCRIPTION1
************************************************************

 01 PROGRAM-NAMES.
    02 PROG1                        PIC X(15)
              VALUE "project-entry  ".
    02 PROG2                        PIC X(15)
              VALUE "look-up        ".
    02 PROG3                        PIC X(15)
              VALUE "project-info   ".
    02 PROG4                        PIC X(15)
              VALUE "event-detail   ".
    02 PROG5                        PIC X(15)
              VALUE "event-revised  ".
    02 PROG6                        PIC X(15)
              VALUE "               ".
    02 PROG7                        PIC X(15)
              VALUE "               ".
    02 PROG8                        PIC X(15)
              VALUE "               ".
    02 PROG9                        PIC X(15)
              VALUE "               ".
    02 PROG10                       PIC X(15)
              VALUE "               ".
    02 PROG11                       PIC X(15)
              VALUE "               ".
```

Figure 12-26. SCREEN COBOL Source Code for Sample Menu Program
(Continued)

```
      02 PROG12                   PIC X(15)
                 VALUE "               ".
      02 PROG13                   PIC X(15)
                VALUE "               ".
      02 PROG14                   PIC X(15)
                VALUE "               ".
      02 PROG15                   PIC X(15)
                VALUE "               ".

    01 PROGRAM-CHOICE REDEFINES PROGRAM-NAMES.
      02 PROGRAM-CHOICE-VALUE
                 OCCURS 15 TIMES PIC X(15).
   ************************************************************
   *   PLACE THE DESCRIPTIONS OF THE CALLED PROGRAMS IN
   *   THE VALUE CLAUSES BELOW.  DESCRIPTION-1 CORRESPONDS
   *   TO FUNCTION KEY 1 AND PROG1.  LEAVE THE DESCRIPTIONS
   *   BLANK FOR THE FUNCTION KEYS NOT USED.  THE PROGRAM
   *   LOGIC INSURES CORRECT DISPLAY AND FUNCTION KEY
   *   CONTROL FOR THOSE OPTIONS WITH NO DESCRIPTIONS.
   ************************************************************

    01 PROGRAM-DESCRIPTIONS.
      02 DESCRIPTION-1            PIC X(40)
              VALUE "To enter a new project.                 ".
      02 DESCRIPTION-2            PIC X(40)
              VALUE "To look up an employee number.          ".
      02 DESCRIPTION-3            PIC X(40)
              VALUE "To obtain information about a project.  ".
      02 DESCRIPTION-4            PIC X(40)
              VALUE "To obtain detailed event information.   ".
      02 DESCRIPTION-5            PIC X(40)
              VALUE "To revise an event.                     ".
      02 DESCRIPTION-6            PIC X(40)
              VALUE "                                        ".
      02 DESCRIPTION-7            PIC X(40)
              VALUE "                                        ".
      02 DESCRIPTION-8            PIC X(40)
              VALUE "                                        ".
      02 DESCRIPTION-9            PIC X(40)
              VALUE "                                        ".
      02 DESCRIPTION-10           PIC X(40)
              VALUE "                                        ".
```

Figure 12-26. SCREEN COBOL Source Code for Sample Menu Program (Continued)

```
     02 DESCRIPTION-11            PIC X(40)
             VALUE "                                        ".
     02 DESCRIPTION-12            PIC X(40)
             VALUE "                                        ".
     02 DESCRIPTION-13            PIC X(40)
             VALUE "                                        ".
     02 DESCRIPTION-14            PIC X(40)
             VALUE "   There are no more options.           ".
     02 DESCRIPTION-15            PIC X(40)
             VALUE "                                        ".

   01 DESCRIPTION-VALUES REDEFINES PROGRAM-DESCRIPTIONS.
     02 DESCRIPTION-ELEMENT
               OCCURS 15 TIMES PIC X(40).
   01 APPLICATION-TITLE.
  *
  *  Insert the screen title in the following VALUE clause.
  *
     02 APPLICATION-NAME          PIC X(60)
     VALUE "PROJECT TRACKING
  -    "          ".

   01 FUNCTION-KEY-SCREEN.
     02 DATA-F1                   PIC X(4).
     02 DATA-F2                   PIC X(4).
     02 DATA-F3                   PIC X(4).
     02 DATA-F4                   PIC X(4).
     02 DATA-F5                   PIC X(4).
     02 DATA-F6                   PIC X(4).
     02 DATA-F7                   PIC X(4).
     02 DATA-F8                   PIC X(4).
     02 DATA-F9                   PIC X(4).
     02 DATA-F10                  PIC X(4).
     02 DATA-F11                  PIC X(4).
     02 DATA-F12                  PIC X(4).
     02 DATA-F13                  PIC X(4).
     02 DATA-F14                  PIC X(4).
     02 DATA-F15                  PIC X(4).

   01 FUNCTION-KEY-SARRAY REDEFINES FUNCTION-KEY-SCREEN.
     02 SARRAY-ELEMENT
            OCCURS 15 TIMES PIC X(4).
```

Figure 12-26. SCREEN COBOL Source Code for Sample Menu Program (Continued)

```
 01 FUNCTION-KEY-NAMES.
   02 K-DATA-F1                   PIC X(4) VALUE "  F1".
   02 K-DATA-F2                   PIC X(4) VALUE "  F2".
   02 K-DATA-F3                   PIC X(4) VALUE "  F3".
   02 K-DATA-F4                   PIC X(4) VALUE "  F4".
   02 K-DATA-F5                   PIC X(4) VALUE "  F5".
   02 K-DATA-F6                   PIC X(4) VALUE "  F6".
   02 K-DATA-F7                   PIC X(4) VALUE "  F7".
   02 K-DATA-F8                   PIC X(4) VALUE "  F8".
   02 K-DATA-F9                   PIC X(4) VALUE "  F9".
   02 K-DATA-F10                  PIC X(4) VALUE " F10".
   02 K-DATA-F11                  PIC X(4) VALUE " F11".
   02 K-DATA-F12                  PIC X(4) VALUE " F12".
   02 K-DATA-F13                  PIC X(4) VALUE " F13".
   02 K-DATA-F14                  PIC X(4) VALUE " F14".
   02 K-DATA-F15                  PIC X(4) VALUE " F15".

 01 FUNCTION-KEY-NARRAY REDEFINES FUNCTION-KEY-NAMES.
   02 NARRAY-ELEMENT
        OCCURS 15 TIMES PIC X(4).

  01 GLOBALS.
    05 SCRN-MESSAGE                PIC X(78) VALUE spaces.
    05 SCRN-MESSAGE-FLDS redefines SCRN-MESSAGE.
      10 MESSAGE-1-NUMBER         PIC S9999
         SIGN IS LEADING SEPARATE CHARACTER.
      10 FILLER                   PIC XX.
      10 MESSAGE-2-NUMBER         PIC S9999
         SIGN IS LEADING SEPARATE CHARACTER.
      10 FILLER                   PIC XX.
      10 MESSAGE-ALPHA.
         15 MESSAGE-ALPHA-A       PIC X(38).
         15 MESSAGE-ALPHA-B       PIC X(26).

 SCREEN SECTION.

 01 MSCREEN1 OVERLAY SIZE 24, 80 .
   05 FILLER AT 1, 2
          VALUE "===========================================
-      "====================================" .
```

Figure 12-26. SCREEN COBOL Source Code for Sample Menu Program (Continued)

```
     05 SCREEN-APPLICATION AT 2, 3
                         PIC X(60)
                         FROM APPLICATION-NAME .
     05 FILLER AT 3, 3
               VALUE "MASTER MENU" .
     05 FILLER AT 4, 2
            VALUE
       "======================================================
  -      "========================".
     05 FILLER AT 5,2
            VALUE "Select one of the following or press SF16
  -           "to EXIT" .
     05 FILLER AT 6, 6
               VALUE "Function Key" .
     05 FILLER AT 6, 33
               VALUE "Description" .
     05 FILLER AT 7, 6
               VALUE "-----------   --------------------------
  -              "---------------" .
     05 K-F1 AT 8, 10
             PIC X(4)
             FROM DATA-F1 .
     05 SCREEN-DESC1 AT 8, 21
                     PIC X(40)
                     FROM DESCRIPTION-1 .
     05 SCREEN-F2 AT 9, 10
                  PIC X(4)
                  FROM DATA-F2 .
     05 SCREEN-DESC2 AT 9, 21
                     PIC X(40)
                     FROM DESCRIPTION-2 .
     05 SCREEN-F3 AT 10, 10
                  PIC X(4)
                  FROM DATA-F3 .
     05 SCREEN-DESC3 AT 10, 21
                     PIC X(40)
                     FROM DESCRIPTION-3 .
     05 SCREEN-F4 AT 11, 10
               PIC X(4)
               FROM DATA-F4 .
```

Figure 12-26. SCREEN COBOL Source Code for Sample Menu Program (Continued)

```
     05 SCREEN-DESC4 AT 11, 21
                     PIC X(40)
                     FROM DESCRIPTION-4 .
     05 SCREEN-F5 AT 12, 10
                  PIC X(4)
                  FROM DATA-F5 .
     05 SCREEN-DESC5 AT 12, 21
                     PIC X(40)
                     FROM DESCRIPTION-5 .
     05 SCREEN-F6 AT 13, 10
                  PIC X(4)
                  FROM DATA-F6 .
     05 SCREEN-DESC6 AT 13, 21
                     PIC X(40)
                     FROM DESCRIPTION-6 .
     05 SCREEN-F7 AT 14, 10
                  PIC X(4)
                  FROM DATA-F7 .
     05 SCREEN-DESC7 AT 14, 21
                     PIC X(40)
                     FROM DESCRIPTION-7 .
     05 SCREEN-F8 AT 15, 10
                  PIC X(4)
                  FROM DATA-F8 .
     05 SCREEN-DESC8 AT 15, 21
                     PIC X(40)
                     FROM DESCRIPTION-8 .
     05 SCREEN-F9 AT 16, 10
                     PIC X(4)
                     FROM DATA-F9 .
     05 SCREEN-DESC9 AT 16, 21
                     PIC X(40)
                     FROM DESCRIPTION-9 .
     05 SCREEN-F10 AT 17, 10
                   PIC X(4)
                   FROM DATA-F10 .
     05 SCREEN-DESC10 AT 17, 21
                      PIC X(40)
                      FROM DESCRIPTION-10 .
```

Figure 12-26. SCREEN COBOL Source Code for Sample Menu Program (Continued)

```
      05 SCREEN-F11 AT 18, 10
                    PIC X(4)
                    FROM DATA-F11 .
      05 SCREEN-DESC11 AT 18, 21
                       PIC X(40)
                       FROM DESCRIPTION-11 .
      05 SCREEN-F12 AT 19, 10
                    PIC X(4)
                    FROM DATA-F12 .
      05 SCREEN-DESC12 AT 19, 21
                       PIC X(40)
                       FROM DESCRIPTION-12 .
      05 SCREEN-F13 AT 20, 10
                    PIC X(4)
                    FROM DATA-F13 .
      05 SCREEN-DESC13 AT 20, 21
                       PIC X(40)
                       FROM DESCRIPTION-13 .
      05 SCREEN-F14 AT 21, 10
                    PIC X(4)
                    FROM DATA-F14 .
      05 SCREEN-DESC14 AT 21, 21
                       PIC X(40)
                       FROM DESCRIPTION-14 .
      05 SCREEN-F15 AT 22, 10
                    PIC X(4)
                    FROM DATA-F15 .
      05 SCREEN-DESC15 AT 22, 21
                       PIC X(40)
                       FROM DESCRIPTION-15 .
      05 FILLER AT 23, 2
      VALUE "==================================================
   -          "=========================".
      05 MESS-AGE AT 24, 1
                    PIC X(79)
                    FROM SCRN-MESSAGE
                    ADVISORY
                    UPSHIFT OUTPUT .
```

Figure 12-26. SCREEN COBOL Source Code for Sample Menu Program
(Continued)

```
**************************************************************
 PROCEDURE DIVISION.
**************************************************************
 DECLARATIVES.
   RECOVER-MAIN-SCREEN SECTION.
     USE FOR SCREEN RECOVERY ON MSCREEN1.
       MOVE SPACES TO SCRN-MESSAGE.
       MOVE "SCREEN RECOVERY" TO MESSAGE-ALPHA-A.
       DISPLAY MSCREEN1.
      DISPLAY TEMP MESS-AGE.
 END DECLARATIVES.

 MAIN SECTION.
   PERFORM 100-INITIALIZATION.
   PERFORM 200-READ-SCREEN until end-program = "Y".
   PERFORM END-OF-PROGRAM.

 END-OF-PROGRAM.
   EXIT PROGRAM.

 100-INITIALIZATION.
*  based on the contents of the description-x fields,
*  determine how many options there are and set MAX-CHOICE.
     PERFORM 110-GET-MAX VARYING NCOUNT
     FROM 1 BY 1 UNTIL TASK-DONE.
     DISPLAY BASE MSCREEN1.
     DISPLAY MSCREEN1.
 110-GET-MAX.
   IF PROGRAM-CHOICE-VALUE(NCOUNT) IS NOT EQUAL TO SPACES
       MOVE NCOUNT TO MAX-CHOICE
       MOVE NARRAY-ELEMENT(NCOUNT) TO SARRAY-ELEMENT(NCOUNT)
   ELSE
       NEXT SENTENCE.
   IF PROGRAM-CHOICE-VALUE(NCOUNT) IS EQUAL TO SPACES
       MOVE "Y" TO TASK.
 200-READ-SCREEN.
   PERFORM 300-ACCEPT-SCREEN.
   IF OPERATION-CHOICE IS EQUAL TO 17
       MOVE "Y" TO END-PROGRAM
   ELSE
```

Figure 12-26. SCREEN COBOL Source Code for Sample Menu Program
(Continued)

```
         IF (OPERATION-CHOICE IS GREATER THAN MAX-CHOICE OR
              OPERATION-CHOICE IS EQUAL TO 16)
            MOVE SPACES TO SCRN-MESSAGE
            MOVE "THIS FUNCTION KEY NOT ACTIVE"
            TO MESSAGE-ALPHA-A
            DISPLAY TEMP MESS-AGE
         ELSE

            MOVE PROGRAM-CHOICE-VALUE(OPERATION-CHOICE)
              TO PROGRAM-CALLED
           CALL PROGRAM-CALLED
              ON ERROR PERFORM 800-CALL-ERRORS
           DISPLAY BASE MSCREEN1
           DISPLAY MSCREEN1.
       IF (OPERATION-CHOICE IS NOT GREATER THAN MAX-CHOICE)
           DISPLAY BASE MSCREEN1
           DISPLAY MSCREEN1
         ELSE
           NEXT SENTENCE.

    300-ACCEPT-SCREEN.
      ACCEPT MSCREEN1 UNTIL
        F1-KEY
        F2-KEY,
        F3-KEY,
        F4-KEY,
        F5-KEY,
        F6-KEY,
        F7-KEY,
        F8-KEY,
        F9-KEY,
        F10-KEY,
        F11-KEY,
        F12-KEY,
        F13-KEY,
        F14-KEY,
        F15-KEY,
        ESCAPE ON (F16-KEY THRU SF15-KEY),
        SF16-KEY.
```

Figure 12-26. SCREEN COBOL Source Code for Sample Menu Program (Continued)

```
      MOVE TERMINATION-STATUS TO OPERATION-CHOICE.

   800-CALL-ERRORS.
     MOVE SPACES TO SCRN-MESSAGE.
     MOVE TERMINATION-STATUS TO MESSAGE-1-NUMBER.
     MOVE TERMINATION-SUBSTATUS TO MESSAGE-2-NUMBER.
     IF TERMINATION-STATUS IS LESS THAN 31
       MOVE CALL-ERROR-MESSAGE(TERMINATION-STATUS)
         TO MESSAGE-ALPHA-A
     ELSE
       IF TERMINATION-STATUS IS EQUAL TO 44
          MOVE "CALL PROGRAM UNIT NAME IS INVALID"
            TO MESSAGE-ALPHA-A
       ELSE
         MOVE "TERMINATION STATUS ERROR OCCURRED"
            TO MESSAGE-ALPHA-A.
      MOVE PROGRAM-CALLED TO PROG-NAME.
      MOVE ERROR-NOTE TO MESSAGE-ALPHA-A.
```

Figure 12-26. SCREEN COBOL Source Code for Sample Menu Program (Continued)

MODIFYING THE PATHCOM COMMAND FILE TO INTEGRATE THE APPLICATIONS

Figure 12-27 shows the PATHCOM command file created by ENABLE when it generated the project-entry application. This figure also indicates areas of the file that require modification.

```
  SET PATHMON BACKUPCPU 1
  SET PATHWAY MAXTCPS 10
  SET PATHWAY MAXTERMS 10
  SET PATHWAY MAXPROGRAMS 10
  SET PATHWAY MAXSERVERCLASSES 10
  SET PATHWAY MAXSERVERPROCESSES 10
  SET PATHWAY MAXSTARTUPS 10
  SET PATHWAY MAXPATHCOMS 40
  SET PATHWAY MAXASSIGNS 32
  SET PATHWAY MAXPARAMS 32
  START PATHWAY COLD!

  SET TCP PROGRAM $SYSTEM.SYSTEM.PATHTCP2
  SET TCP CPUS 0:1
  SET TCP MAXTERMS 5
  SET TCP MAXSERVERCLASSES 003
  SET TCP MAXSERVERPROCESSES 003
  SET TCP MAXTERMDATA 12036  <----  must be modified to
  SET TCP MAXREPLY    02000         include enough data space
  SET TCP NONSTOP 0                 for all applications
  SET TCP TCLPROG  $DATA.SAMPLE.POBJ
  ADD TCP ENABLE-TCP

  SET PROGRAM TCP ENABLE-TCP
  SET PROGRAM TYPE T16-6520 INITIAL PROJECT-ENTRY <-  must be
  SET PROGRAM TMF OFF                                 changed
  ADD PROGRAM PROJECT-ENTRY  <----------------------  to
                                                      ident-
  RESET SERVER ASSIGN, PARAM                          ify the
                                                      menu
                                                      program
```

Figure 12-27. PATHCOM Command File Before Modifications
(Continued next page)

```
-----------------------------------------------------------------
|                                                               |
|  SET SERVER PROGRAM $SYSTEM.SYSTEM.ENABLEGS                   |
|  SET SERVER CPUS 0:1                                          |
|  SET SERVER NUMSTATIC 1                                       |
|  SET SERVER (ASSIGN MANAGER,EMPLOYS,INPUT)                    |
|  SET SERVER (ASSIGN PROJECTS,PROJECTS)                        |
|  SET SERVER (ASSIGN EVENTS,EVENTS)  <-  additional SET SERVER |
|                                         ASSIGN commands must  |
|  SET SERVER TMF OFF                     be added              |
|  ADD SERVER ENABLE-SERVER                                     |
|                                                               |
-----------------------------------------------------------------
```

Figure 12-27. PATHCOM Command File Before Modifications (Continued)

Figure 12-28 shows the PATHCOM command file, "prfile1," after it has been modified to execute the project-tracking system.

```
-----------------------------------------------------------------
|                                                               |
|  SET PATHMON BACKUPCPU 1                                      |
|  SET PATHWAY MAXTCPS 10                                       |
|  SET PATHWAY MAXTERMS 10                                      |
|  SET PATHWAY MAXPROGRAMS 10                                   |
|  SET PATHWAY MAXSERVERCLASSES 10                              |
|  SET PATHWAY MAXSERVERPROCESSES 10                            |
|  SET PATHWAY MAXSTARTUPS 10                                   |
|  SET PATHWAY MAXPATHCOMS 40                                   |
|  SET PATHWAY MAXASSIGNS 32                                    |
|  SET PATHWAY MAXPARAMS 32                                     |
|  START PATHWAY COLD!                                          |
|                                                               |
-----------------------------------------------------------------
```

Figure 12-28. Sample Modified PATHCOM Command File (Continued next page)

```
SET TCP PROGRAM $SYSTEM.SYSTEM.PATHTCP2
SET TCP CPUS 0:1
SET TCP MAXTERMS 5
SET TCP MAXSERVERCLASSES 003
SET TCP MAXSERVERPROCESSES 003
SET TCP MAXTERMDATA 27732                             (1)
SET TCP MAXREPLY    02000
SET TCP NONSTOP 0
SET TCP TCLPROG  $DATA.PROJ.POBJ
ADD TCP ENABLE-TCP

SET PROGRAM TCP ENABLE-TCP
SET PROGRAM TYPE T16-6520 INITIAL MENU                (2)
SET PROGRAM TMF OFF
ADD PROGRAM MENU                                      (3)

RESET SERVER ASSIGN, PARAM

SET SERVER PROGRAM $SYSTEM.SYSTEM.ENABLEGS
SET SERVER CPUS 0:1
SET SERVER NUMSTATIC 1
SET SERVER (ASSIGN PROJECT-BOX, PROJECTS)             (4)
SET SERVER (ASSIGN PARTIC-BOX, RESPFOR)
SET SERVER (ASSIGN MANAGER, EMPLOYS, INPUT)
SET SERVER (ASSIGN PROJECTS, PROJECTS)
SET SERVER (ASSIGN EVENTS, EVENTS)
SET SERVER (ASSIGN EVENTS-BOX, EVENTS, INPUT)
SET SERVER (ASSIGN EVENT-X, EVENTS, INPUT)
SET SERVER (ASSIGN EVENTS-A, EVENTS, INPUT)
SET SERVER (ASSIGN EMPLOY-ASSIGN, EMPLOYS, INPUT)
SET SERVER (ASSIGN MY-EVENT, EVENTS, INPUT)
SET SERVER (ASSIGN EVENTS-3, EVENTS)
SET SERVER (ASSIGN EVENTS-4, EVENTS)
SET SERVER (ASSIGN EMPLOYEES, EMPLOYS)
SET SERVER (ASSIGN PROJ-BOX, PROJECTS, INPUT)
SET SERVER (ASSIGN EMPLOYEE-PART, RESPFOR, INPUT)
SET SERVER (ASSIGN EVENT-B, EVENTS, INPUT)
SET SERVER (ASSIGN EMPLOY-BOX, EMPLOYS, INPUT)
SET SERVER TMF OFF
ADD SERVER ENABLE-SERVER
```

Figure 12-28. Sample Modified PATHCOM Command File (Continued)

```
  SET TERM FILE $TERM01                                  (5)
  SET TERM TCP ENABLE-TCP
  SET TERM INITIAL MENU
  ADD TERM TERM01

  SET TERM FILE $TERM02                                  (6)
  SET TERM TCP ENABLE-TCP
  SET TERM INITIAL MENU
  ADD TERM TERM02

  SET TERM FILE $TERM03
  SET TERM TCP ENABLE-TCP
  SET TERM INITIAL MENU
  ADD TERM TERM03
```

NOTES

(1) MAXTERMDATA is changed from 12036 (for the "project-entry" application) to 29782. You estimate this value for MAXTERMDATA by using the following DATA SIZE values from the SCREEN COBOL compilation listings of the applications:

Application	DATA SIZE
Menu	359
Project-entry	6018
Employee-assign	4674
Look-up	3840
Project-info	7402
Event-detail	4850
Event-Revised	5160

You use these DATA SIZE values to compute the longest path through the applications as follows:

Figure 12-28. Sample Modified PATHCOM Command File (Continued)

```
Menu to project-entry to employee-assign to
look-up equals:

   359 + 6,018 + 4,674 + 3,840                = 1,4891

Menu to project-info to event-detail equals:

   359 + 7,402 + 4,850                        = 1,2611

Menu to event-detail equals:

   359 + 4,850                                =  5,209

Menu to event-revised equals:

   359 + 5,160                                =  5,519
```

The number of bytes in the longest path is 14,891. Multiplying this figure by 2 gives the estimated value 29,782 for MAXTERMDATA.

(2) You change the INITIAL specification from PROJECT-ENTRY to MENU.

(3) You change the program-name entry from PROJECT-ENTRY to MENU.

(4) You add the SET SERVER ASSIGN commands necessary for all the applications. You can copy these commands from the PATHCOM command files generated for each application.

(5) Optionally, you can add commands to describe a terminal named TERM01 and add a description of this terminal to the PATHWAY system. If you want to execute your applications by entering a PATHCOM RUN PROGRAM command, do not add the TERMINAL commands to your PATHCOM command file.

(6) Optionally, you can add commands to describe and add a terminal named TERM02.

Figure 12-28. Sample Modified PATHCOM Command File (Continued)

SAMPLE OBEY FILES

Figure 12-29 shows a sample obey file that establishes the PATHWAY system.

```
PURGE projlog  <-----------  purges the current PATHCOM

CREATE projlog  <----------  creates a new PATHCOM log file

PATHMON/NAME $one, NOWAIT,
  CPU 0, OUT projlog/  <---  creates a PATHMON process

PATHCOM/IN prfile1/$one  <-  cold starts PATHWAY from
                             information in the named
                             PATHCOM command file

PATHCOM $one; START TCP ENABLE-TCP  <-  starts the TCP

PATHCOM $one; START TERM TERM01  <-  starts execution of a
                                     SCREEN COBOL program
                                     for one terminal in
                                     the PATHWAY system

  Note that if you want to execute your applications by
  entering a PATHCOM RUN PROGRAM command, omit the START
  TERM command from your obey file and create a separate
  obey file that contains the following:

PATHCOM $one;RUN MENU
```

Figure 12-29. Sample Obey File That Establishes the PATHWAY System

Figure 12-30 shows the contents of a sample obey file that suspends the PATHWAY system.

```
PATHCOM $one; SHUTDOWN, WAIT
```

Figure 12-30. Sample Obey File to Suspend the PATHWAY System

APPENDIX A

SYNTAX SUMMARY

This appendix summarizes the syntax of ENABLE commands, attributes and operating commands. A reference to the appropriate page in the ENABLE Reference Manual accompanies each.

```
ENABLE Commands:                                           Page:


   ADD [ APPL ] <object> [ , LIKE <prior-object> ]          3-6
       [ BOX  ]

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


   ASSUME { APPL }                                         3-11
          { BOX  }


   DELETE [ APPL ] { <object-name> }                       3-13
          [ BOX  ] { *             }


   GENERATE [ APPL ] [ <application> ]                     3-15
                     [ *             ]

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


   INFO [ APPL ] { <object> } [ , BRIEF  ]                 3-18
        [ BOX  ] { *        } [ , DETAIL ]
```

```
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
```

```
   SCOBOLSKELETON <file-name>                                   4-53


   SCOBOLSOURCE <file-name> [!]                                 4-55


   SCREENFORMAT { UNCOMPRESSED }                                4-58
                { COMPRESSED   }


   SERVERCLASS <server-class-name>                              4-62


   SIZE <number>                                                4-64


   TERMINAL <terminal-type>                                     4-66


   TITLE <string-literal>                                       4-68


   TMF { ON  }                                                  4-69
       { OFF }


   TREE ( <level> <box-name>                                    4-71

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

       <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>


   UPDATE { ON  }                                              4-102
          { OFF }


   VALUES { ON  }                                              4-103
          { OFF }
```

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 did 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 access-ed 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

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