Tandem NonStop™ and NonStop II™ Systems

ENFORM™ User's Guide

ABSTRACT: This guide provides a task-oriented view of ENFORM for both programmers and nonprogrammers.

PRODUCT VERSION: ENFORM C12

OPERATING SYSTEM VERSION: GUARDIAN A06 (NonStop II System)
GUARDIAN E07 (NonStop System)

Throughout this document, all references to *NonStop II systems* indicate the software that runs on Tandem NonStop II processors and/or NonStop TXP processors.

Tandem Computers Incorporated
19333 Vallco Parkway
Cupertino, California 95014-2599

Manual: Part No. 82349 B00
Update: Part No. 82195

December 1983
Printed in U.S.A.

DOCUMENT HISTORY

Edition	Part Number	Operating System Version	Date
First Edition	82349 A00	GUARDIAN A04/E05	October 1982
Second Edition	82349 B00	GUARDIAN A05/E06	April 1983
Update 1	82195	GUARDIAN A06/E07	December 1983

New editions incorporate all updates issued since the previous edition. Update packages, which are issued between editions, contain additional and replacement pages that you should merge into the most recent edition of the manual.

NEW AND CHANGED INFORMATION

This update to the *ENFORM User's Guide* provides information about the following changes to the ENFORM product:

- ENFORM now allows you to specify more than one LINK OPTIONAL statement in your query specifications. Section 3 contains several examples of query specifications that contain more than one LINK OPTIONAL statement or that combine the LINK OPTIONAL statement with a LINK statement or a WHERE clause.

- Additions to Appendix B and Section 6 describe new ENFORM error messages.

- Miscellaneous technical and editorial changes have been made.

All changes from the last edition are marked with a revision bar in the margin. Each page containing a change has the "December 1983" date at the bottom of the page.

CONTENTS

FIGURES

TABLES

PREFACE

This guide is one of three volumes that describe ENFORM. This guide provides guidelines to follow when you:

- develop a data base from which you want to produce ENFORM reports
- write ENFORM queries
- attempt to improve the performance of your ENFORM queries
- use the Host Language interface
- write an ENFORM server.

The intended audience for this guide is any person who is familiar with a Tandem system. For more information about ENFORM and related products, refer to the publications listed below.

Data Definition Language (DDL) Programming Manual

EDIT Manual

ENFORM Reference Manual

ENSCRIBE Programming Manual

GUARDIAN Operating System Command Language and Utilities Manual

GUARDIAN Operating System Programming Manual

Introduction to ENFORM

SYNTAX CONVENTIONS IN THIS MANUAL

This table describes the characters and symbols used in this manual's syntax notation. For distinction, syntactical elements appear in a typeface different from that of ordinary text.

Notation	Meaning
UPPERCASE LETTERS	All keywords and reserved words appear in capital letters. If a keyword can be abbreviated, the part that can be omitted is enclosed in brackets.
lowercase letters	All variable entries supplied by the user are shown in lower-case characters.
Brackets	Square brackets ([ ]) enclose all optional syntax elements. A vertically-aligned group of elements enclosed in brackets represents a list of selections from which to choose one or none.
Braces	A vertically-aligned group of syntax elements enclosed in braces ({ }) represents a list of selections from which exactly one must be chosen.
Ellipsis	When an ellipsis (...) immediately follows a pair of brackets or a pair of braces, the enclosed syntax can be repeated any number of times.
Punctuation	Parentheses, commas, and other punctuation or symbols 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 must actually be entered.

SECTION 1

INTRODUCTION

ENFORM, a product that is part of the ENCOMPASS Distributed Data Base Management System, enables you to simply and efficiently:

- Retrieve data from a data base.
- Perform calculations (such as addition, subtraction, multiplication, and division) upon the retrieved data.
- Sort and group the retrieved data.
- Perform cumulative operations (such as counting, totaling, and averaging) upon the retrieved data.
- Format and print a report containing the retrieved data.
- Create a new physical file containing the retrieved data.

The ENFORM language consists of statements, clauses, and commands that you use to provide ENFORM with a detailed description (called a query specification) of: the data to be retrieved from the data base, the operations to be performed on that data, and the form in which the data is to be returned to you.

This guide discusses the use of ENFORM language elements but does not discuss the syntax that you use to specify these elements. To obtain a brief summary of the syntax enter the ENFORM ?HELP command or refer to Appendix A. For detailed syntax information, refer to the *ENFORM Reference Manual*.

USING ENFORM—OVERVIEW

Before ENFORM can retrieve any data, you must perform tasks that provide ENFORM with the environment and information it needs to retrieve the data. Figure 1-1 shows these tasks.

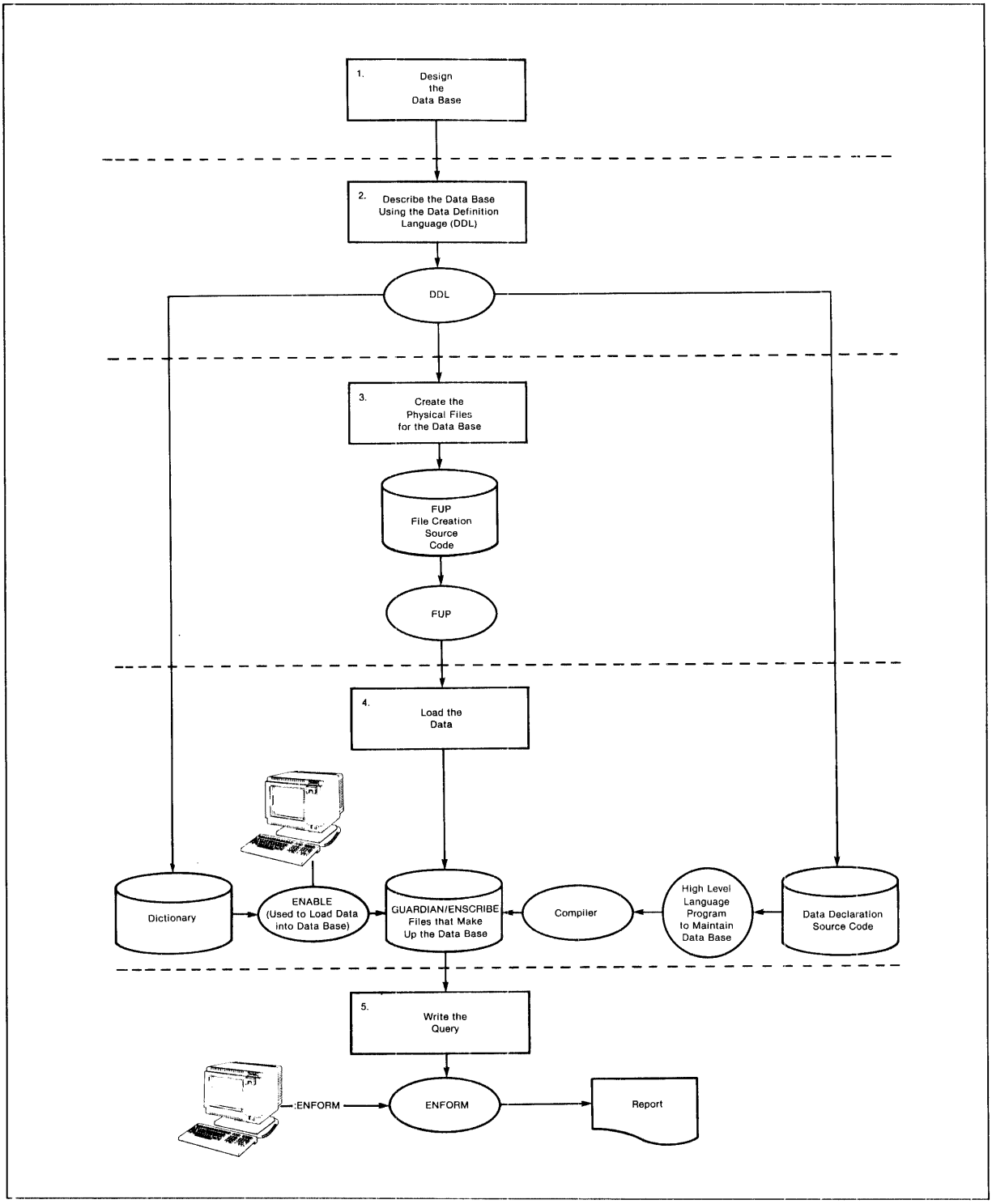


Figure 1-1. Overview of Tasks Involved in Using ENFORM

The following paragraphs provide a brief overview of the tasks shown in Figure 1-1 (refer to Sections 2, 3, and 4 for detailed information). The tasks are organized into steps as follows:

1. Design the data base. This guide does not describe a process for data base design. Section 2 does, however, discuss some basic guidelines for developing or producing a data base to be used with ENFORM.

2. Use the Data Definition Language (DDL) to describe the data. Submit the DDL source code to the DDL compiler, which produces source code for the File Utility program (FUP) and the data declaration sections of COBOL, FORTRAN, and TAL programs. The DDL compiler also produces a dictionary that provides a description of the data to all the applications using the data base.

3. Use FUP to process the source code from the previous step and to create the physical GUARDIAN/ENSCRIBE files that store the data.

4. Choose a method of loading the data. In Figure 1-1, an ENABLE-generated application is used. ENABLE accesses the dictionary to extract record descriptions of the data; the application it creates allows you to load the data into the data base. Alternatively, you could use the data declaration source code created by the DDL compiler to produce a high level language program that loads the data.

5. Use the ENFORM language elements to write requests, called queries, that retrieve data from the data base and format the retrieved data into a report.

ENFORM PROCESSING ENVIRONMENT

The environment in which ENFORM processes data consists of several components. These components can be divided into three categories:

1. The required user-supplied components: a dictionary, a data base, and a query specification. To supply these components, complete the tasks described earlier in this section.

2. The ENFORM processes: the query compiler/report writer and the query processor. Tandem supplies these processes with ENFORM.

3. The optional user-supplied components: a host language program and an ENFORM server. (A host language program allows you to use ENFORM to access data through the host language interface. An ENFORM server allows you to use ENFORM to access data that ENFORM would normally find unusable.)

The following paragraphs briefly describe these components and the role that each plays in the ENFORM processing environment.

Figure 1-2 shows the ENFORM processes and the required user-supplied components of the ENFORM processing environment.

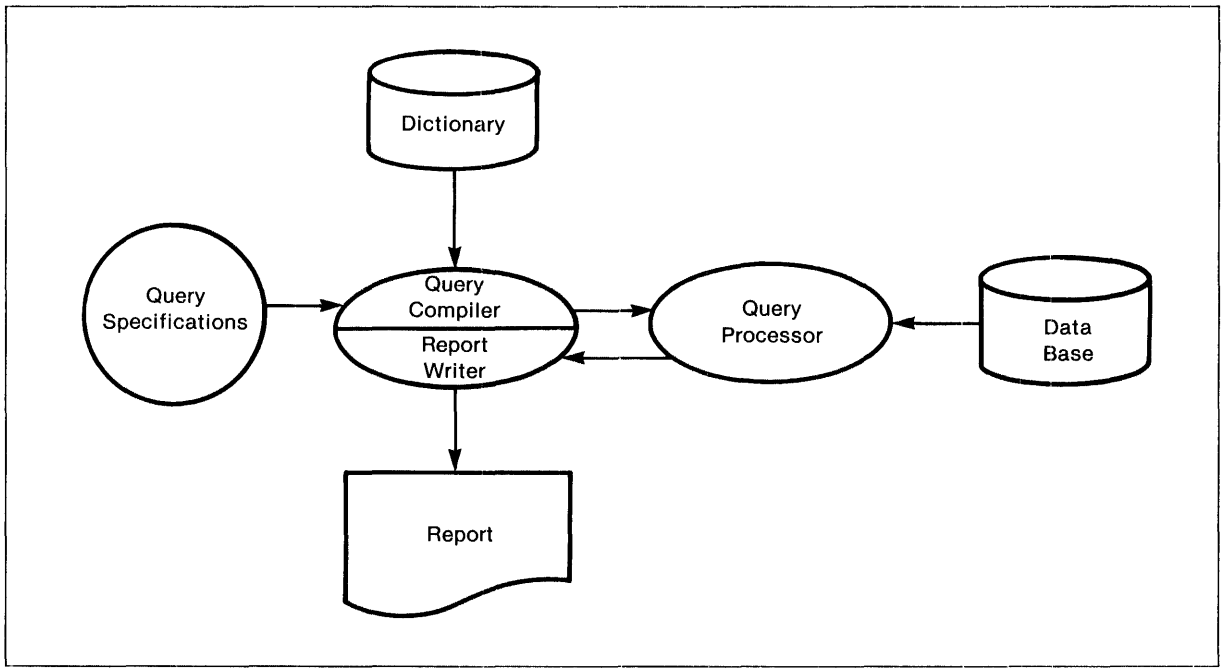


Figure 1-2. ENFORM Processing Environment

The Dictionary

The dictionary is a collection of seven files that define the contents of a data base. The dictionary provides ENFORM with a complete description of the structure of each record in the data base. Before retrieving any data, ENFORM accesses the dictionary, obtains information about the record descriptions, and stores this information in the internal table of the query compiler/report writer. Refer to Section 2 for more information about the dictionary.

The Data Base

The data base consists of the physical files that contain the actual data. By using the information obtained from the dictionary, ENFORM searches the data base to find the data you want. After finding the data, ENFORM returns the data to the query processor. Refer to Section 2 for more information about the components of a data base.

A Query Specification

A query specification (which consists of ENFORM statements, clauses, or commands) is a detailed description of the information you want to retrieve from the data base. ENFORM uses the information provided by the query specification to determine the query environment, the data to be retrieved, and the form in which this data is to be returned to you.

Refer to Section 3 for more information about using the ENFORM statements, clauses, and commands to write a query specification.

The Query Compiler/Report Writer

The query compiler/report writer has two logical functions or phases: (1) compiling the query and (2) formatting and writing a report.

During the first phase, the query compiler checks the query specifications for syntactical correctness. If the query compiler detects errors at this time, query processing is stopped and a syntax error message is issued.

If the query specifications are correct, the query compiler compiles the query specifications into a form (called a compiled representation of the query) that can be understood by the query processor. The compiled representation of the query includes information from the dictionary about the records and fields from which data is to be retrieved. It also contains information about qualifications in the query specifications that limit the number of records read from the data base and written to the target records (the temporary records built by the query processor from which your ENFORM output is produced). The query compiler sends the compiled query representation to the query processor as an interprocess message.

During the first (or compilation) phase, the query compiler also builds an internal report specification that is used by the report writer during the second (or report-writing) phase.

The second phase occurs only if the query specifications contain a request for a report (the presence of a LIST statement within the query specifications requests a report). During this phase, the report writer reads each target record returned by the query processor, formats it using information from the internal report specification built during the first phase, and produces the report.

The Query Processor

The query processor is a server process that receives query specifications and record description information from the query compiler. After receiving this information, the query processor retrieves data from the data base. The query processor exists in one of the following forms:

1. As a dedicated server process that is created for and provides services to an individual query compiler/report writer process. This dedicated server process is created each time an ENFORM session is initiated. One dedicated query processor executes all the ENFORM queries within the session.

2. As a named server process that can be shared (sequentially) by several ENFORM queries. This type of process exists until it stops or times out, has a backup process, and handles queries one at a time.

The query processor automatically chooses a strategy for accessing your data base by using information from the dictionary and the file system. It uses its own search algorithms to reduce the number of data base accesses needed to produce the specified data. The query processor produces target records containing all the data requested in the query specifications. The query processor places the target records in a temporary work file called a target file.

After producing the target records, the query processor does one of the following:

- Returns the target records to the compiler/report writer if the query specifications contained a LIST statement. The query processor returns the target records as a unit in the target file if they required sorting or if more than one data base record was accessed. If the target records do not require sorting and only one data base record is accessed, the query processor returns the target records directly.

- Renames the target file as a FIND file (with an unstructured file type) when the query specifications include a FIND statement. The dictionary must contain a description of the FIND file.

- Transmits the target records one by one to a host language program.

Host Language Interface

Figure 1-3 shows the role of a host language program in the ENFORM processing environment.

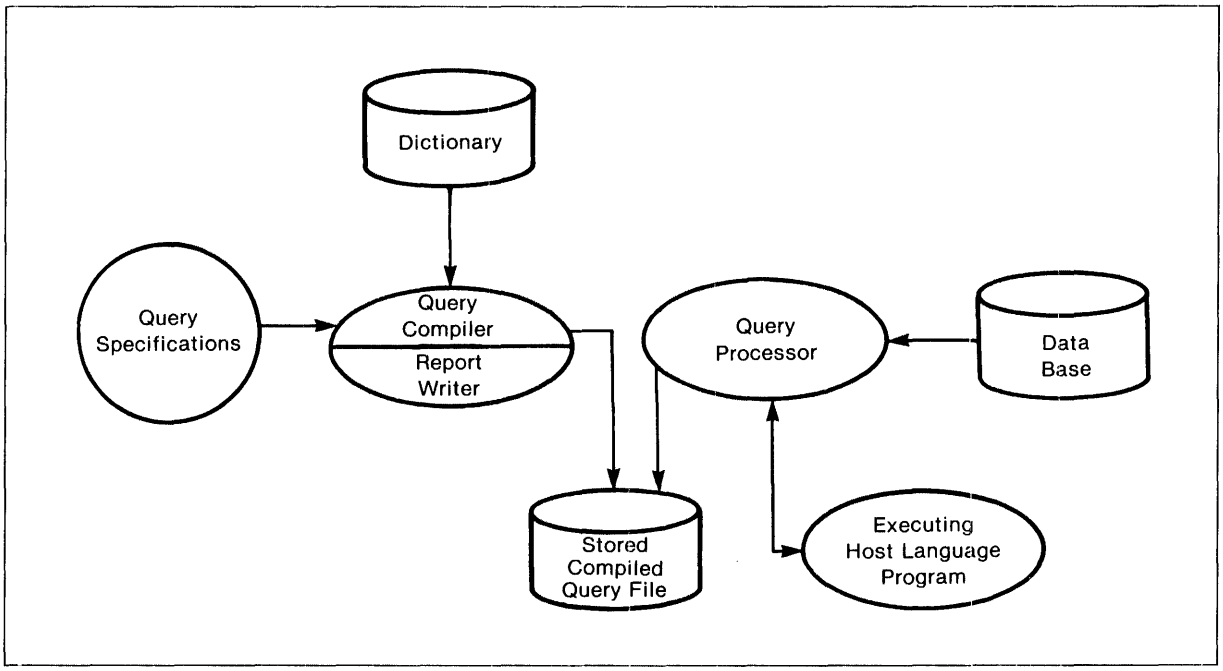


Figure 1-3. Host Language Program in the ENFORM Processing Enviroment

A host language program allows you to use ENFORM to retrieve data from a data base and then perform operations on the data that are not possible when ENFORM is used alone. Before a host language program executes, the query compiler must compile and save your query specifications (containing a FIND statement) in a compiled query file. By including a series of ENFORM procedures, the host language program supplies the query processor with the name of the compiled query file and the value of any included parameters. The query processor retrieves the information specified and returns the information to the host language program, one target record at a time.

Refer to Section 6 for more information about the host language interface.

ENFORM Server

Figure 1-4 shows the role of an ENFORM server (process file) in the ENFORM processing environment.

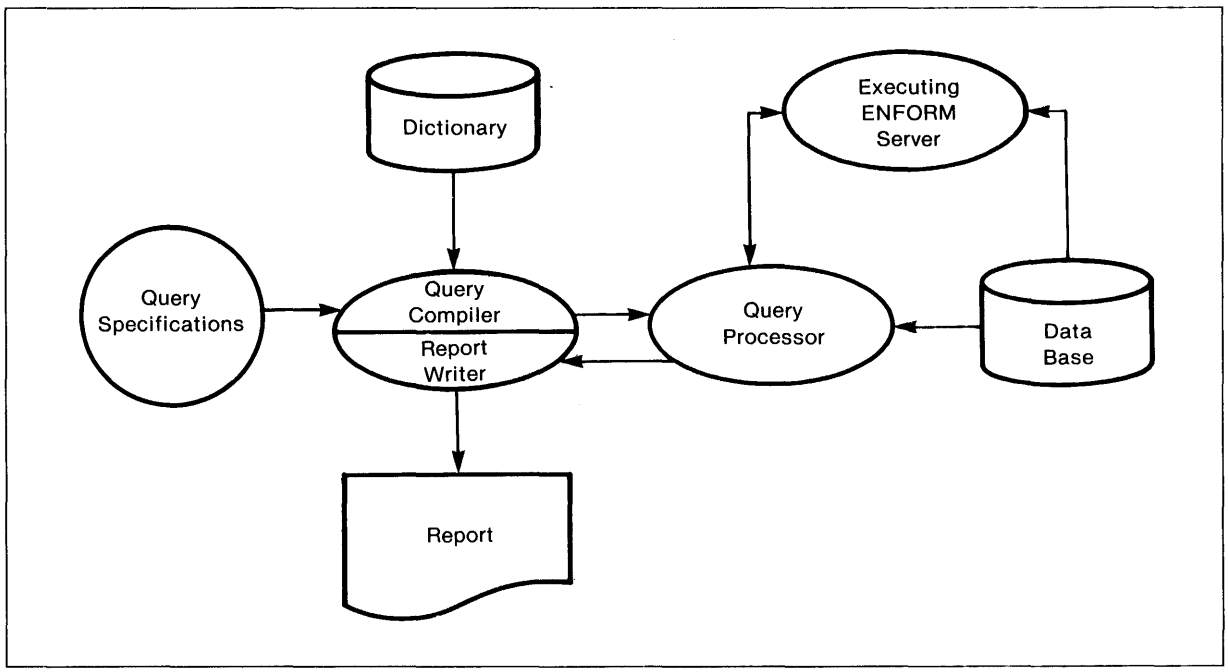


Figure 1-4. ENFORM Server in the ENFORM Processing Environment

An ENFORM server is a user-written process file that extends the query capability of ENFORM by allowing the query processor to use data that might otherwise be unusable. An ENFORM server appears to the query processor as a data file. When an ENFORM server receives a request from the query processor for a data record, the server replies by either returning the data record to the query processor or by indicating there are no more records. An ENFORM server terminates when a pair of messages pass between it and the query processor stating that no more requests exist.

Refer to Section 7 for more information about an ENFORM server.

ENFORM TERMINOLOGY

This guide uses the following terms to discuss the ENFORM language and the output produced by ENFORM:

- Query specifications

  the language elements (statements, clauses, commands, ...) that you specify to provide ENFORM with the information it needs to retrieve data and to establish the query environment.

- Query

  one complete LIST or FIND statement. Both the LIST and the FIND statements specify the information to be retrieved.

- Target-list

  a part of the query; a target-list is separated into target-items and by-items.

  Target-items

  any record names, field names, variable names, aggregate names, literals, or expressions, including by-items, whose values you want to appear as output from your query.

  By-items

  field names modified by a BY or BY DESC clause. The values of the fields are used to sort and group the query output.

- Request-qualification

  a condition or conditions that a data base element must meet before it is selected to contribute to your query output. A request-qualification begins with a WHERE clause followed by a logical expression.

- Target-records

  the temporary records built by the query processor from which your ENFORM output is produced. The query processor returns the target-records to the query compiler/report writer when the output is to be formatted and written as a report.

SECTION 2

DEVELOPING THE DATA BASE

ENFORM enables you to retrieve data from a data base. This section discusses some of the tasks involved in developing or producing a data base. It does not recommend a method of data base design. Data base design is a complex task, one that is beyond the scope of this guide.

Before performing the tasks described in this section, it is important to understand the characteristics of the data that comprise a data base.

WHAT IS A DATA BASE?

A data base is a collection of data that is stored and used for multiple purposes. Usually many different kinds of applications access a data base that contains many different types of data. Thus, a data base serves as a repository for the data needed to perform certain functions in a commercial, scientific, or business enterprise.

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 two data categories: either alphanumeric or numeric. Alphanumeric fields contain data composed of letters of the alphabet, spaces, digits, and special symbols like the hyphen. Numeric fields contain digits, minus or plus signs, and decimal points.

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 group.

Records

A record (sometimes called a record-type) is a collection of associated fields. Each record has a name. Consider Figure 2-1 which shows the records named *parts* and *odetail.*

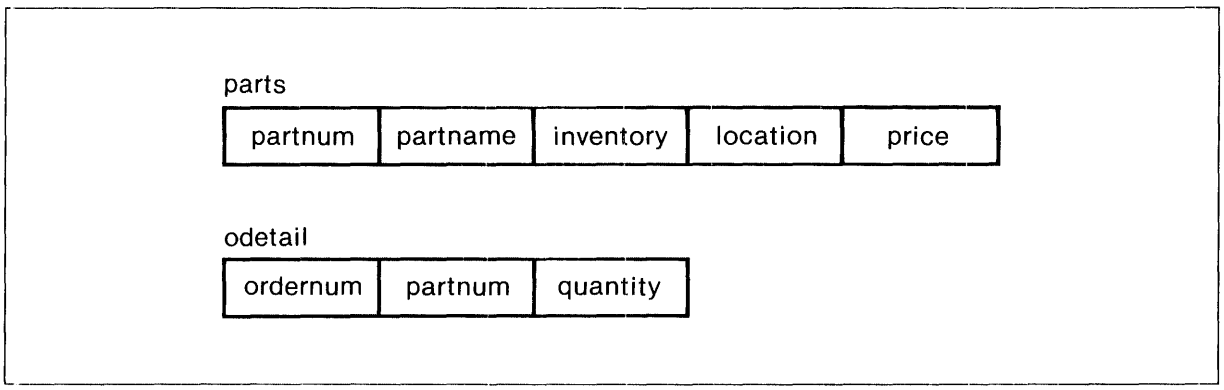


Figure 2-1. Sample Records

The record *parts* consists of the fields: *partnum, partname, inventory, location,* and *price.* The record *odetail* consists of the fields *ordernum, partnum,* and *quantity.*

No field values are associated with a record; instead a record acts as a framework into which specific field values can be fitted. Usually many different types of records exist within a data base, with each different type of record having its own set of record occurrences.

Record Occurrences

A record occurrence contains the actual data which is retrieved by an application program. For example, consider Figure 2-2 which shows some record occurrences for both *parts* and *odetail.*

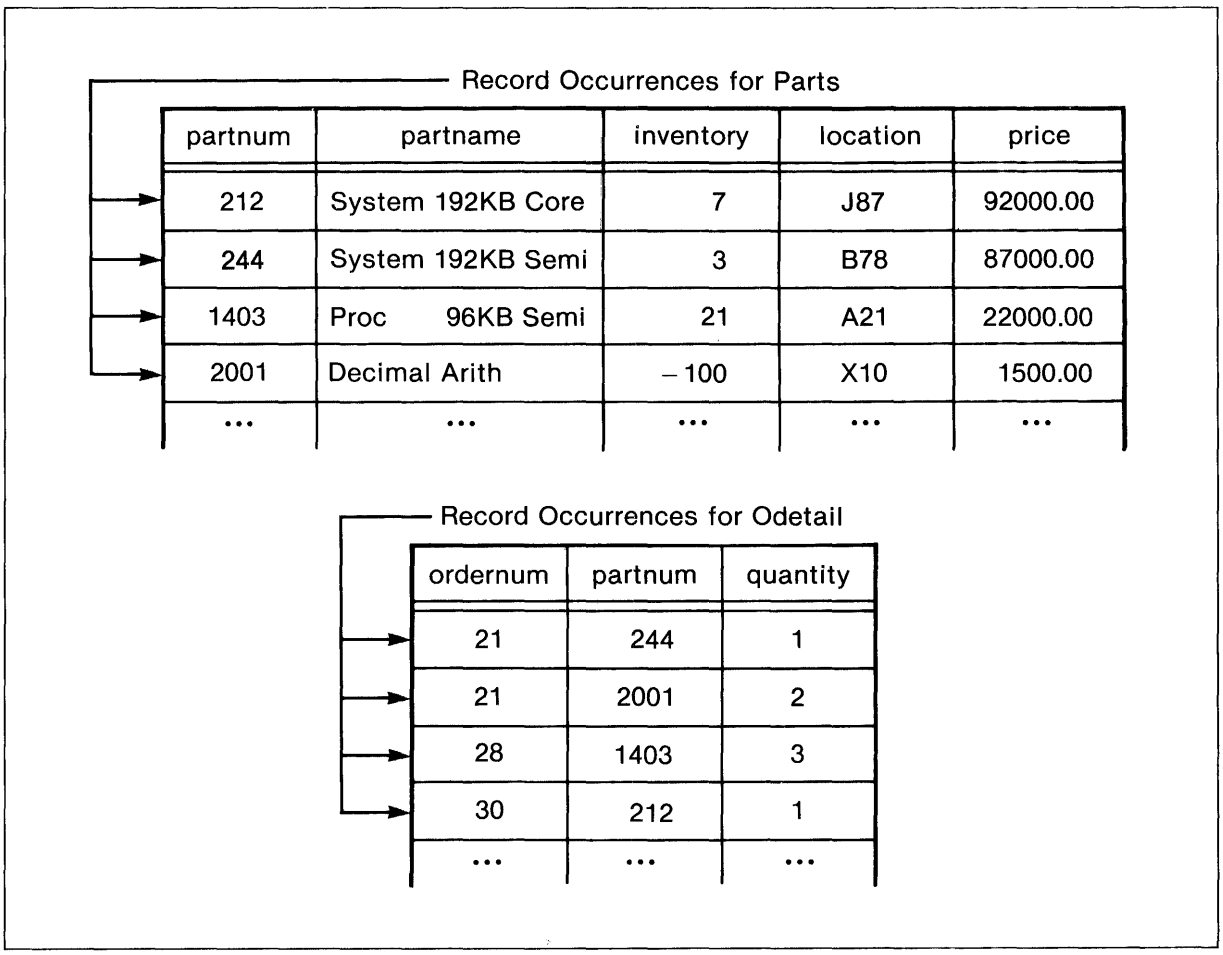


Figure 2-2. Sample Record Occurrences

Each *parts* record occurrence contains the field values that concern a specific part. Each *odetail* record occurrence contains the field values that concern a specific order.

Notice that both *parts* and *odetail* contain the same field values for *partnum.* When the same field values occur in different sets of record occurrences, you can easily establish a connecting relationship between the record occurrences (even if the fields have different names). When you establish such a connecting relationship, the process is called linking. Refer to Section 3 for more information about linking.

Key Fields

A key field is a field whose value an application can use to identify a specific record occurrence. Sometimes more than one field is needed to identify a record. When two or more contiguous fields are used to identify a record occurrence, the combined fields are called composite key fields. The two categories of key fields are primary key fields and alternate key fields.

A primary key field is a field whose value an application uses to uniquely identify a particular record occurrence. Only one primary key field can exist for a record. A likely primary key field for *parts* is *partnum* because this field uniquely identifies each record occurrence of *parts*. *Odetail* requires a composite key composed of both *ordernum* and *partnum* because the same part number could occur in many orders and the same order number could contain many part numbers.

An alternate key field is a field whose value an application uses to identify all record occurrences with a certain property. A record can have more than one alternate key. When an alternate key is specified for a record, an alternate search path exists that can be used to retrieve data. The presence of alternate keys affects any application that uses the data base. Refer to Section 5 for more information about alternate key fields.

TASKS INVOLVED IN DEVELOPING YOUR DATA BASE

The following discussion describes the tasks involved in developing your data base. The tasks are:

- Normalizing the data.
- Describing the data base.
- Creating the physical files that store the data.
- Loading the data.

Normalizing the Data

ENFORM (and most other applications that access the data base) will retrieve data most efficiently when record occurrences do not contain repeating groups. Remember, a repeating group is a field that stores more than one value.

Use the process of normalization to remove the repeating groups from your data. Consider Figure 2-3 which shows the record occurrences of *oldorder*. In *oldorder* both *partnum* and *quantity* are repeating groups.

Oldorder Record Occurrences

ordernum	partnum	ordate	deldate	quantity	custnum
21	244 2001 2403 4103	011078	041078	1 2 2 2	1234
25	244 5103 6301 6402	012378	061578	1 2 1 10	7777
....	...	...	...	...	...

Figure 2-3. Record Occurrences With Unnormalized Data

The process of normalization removes the repeating groups. Four levels of normalization exist. This guide discusses only the first level of normalization: first normal form. First normal form exists when the following condition is satisfied:

For every field in a record occurrence there exists precisely one value, never a group of values.

Figure 2-4 shows the record occurrences that result when *oldorder* is normalized in first normal form. The new record is named *neworder.*

Neworder Record Occurrences

ordernum	partnum	ordate	deldate	quantity	custnum
21	244	011078	041078	1	1234
21	2001	011078	041078	2	1234
21	2403	011078	041078	2	1234
21	4103	011078	041078	2	1234
25	244	012378	061578	1	7777
25	5103	012378	061578	2	7777
25	6301	012378	061578	1	7777
25	6402	012378	061578	10	7777
....	...	....	...	...	...

Figure 2-4. Record Occurrences With Normalized Data

First normal form is a sufficient level of normalization for use with ENFORM; however, further normalization might be desirable for other applications using the data base. For example, *neworder* could be split into two records, *order* and *odetail*, as shown in Appendix C.

Although normalization requires that some data values appear in more than one record occurrence, normalization results in a considerable simplification of the data structure. This simplification of the data structure allows ENFORM to operate efficiently on the data base. It also makes the formation of ENFORM queries much simplier.

Describing the Data Base

Before using ENFORM to retrieve data, you must describe the data base fields, records, and files. This description provides ENFORM with information about:

- the name and length of each record.
- the names, locations, lengths, and data categories (either alphanumeric or numeric) of fields within each record.
- the file structure and location of the physical file associated with each record.
- the primary and alternate keys (if any) of each record.

DATA DEFINITION LANGUAGE. Use the Data Definition Language (DDL) to create and manage data descriptions. Using DDL allows you to describe the file, record, and data structures of a data base. DDL creates the dictionary used by ENFORM to obtain information about your data base. DDL also optionally produces file creation source commands for use with the File Utility Program (FUP) and data declaration source code for use with COBOL, FORTRAN, and TAL programs. This guide describes DDL briefly; refer to the *Data Definition Language (DDL) Programming Manual* for detailed information.

DDL statements establish the definitions of data elements in a data base. A DDL RECORD statement defines each record name and includes a DDL FILE IS clause that identifies the Tandem disc file containing the actual data. The RECORD statement also describes the record structure including field names, data categories, and optionally a heading and display format for each field. If a record has key fields, the RECORD statement defines the fields that are primary and alternate key fields.

Figure 2-5 shows an example RECORD statement which describes the record named *parts*. The FILE IS clause identifies the file storing the data as *$mkt.sample.parts* and indicates that the file type is key-sequenced. The RECORD statement identifies the fields named *partnum*, *inventory*, *location*, and *price*. All of these fields are numeric with the exception of *partname* and *location*. (A numeric field is described as PIC 9...; an alphanumeric field is described as PIC X...) The record description describes the primary key as *partnum* and the alternate key as *partname*.

```
RECORD parts.
        FILE IS $mkt.sample.parts       KEY-SEQUENCED.
  02    partnum                         PIC 9(4).
  02    partname                        PIC X(18).
  02    inventory                       PIC 999S.
  02    location                        PIC XXX.
  02    price                           PIC 999999V99.
KEY IS partnum.
KEY "pn" IS partname.
END
```

Figure 2-5. Sample DDL Record Description

When you use DDL to describe data, remember the following:

- Associate only one record with each physical file with one exception: unstructured FIND files can be associated with more than one record.
- Describe only associated information in each record.
- Avoid repeating groups in record descriptions. A record description with an OCCURS clause causes a repeating group. If necessary, divide the file into two or more files.
- Use the SEQUENCE IS clause for non-key-sequenced files if the records are already sorted according to the value of a field. When the field is modifed by a BY or ASCD clause, ENFORM reads the SEQUENCE IS clause and suppresses its own sort process thus reducing processing time.
- Specify the actual file type of the physical file (or files) associated with the record description. ENFORM obtains the file type from the dictionary. If the file type of a physical file is not the same as the file type specified in the dictionary, your query might return unexpected results.

DATA DICTIONARY. The data dictionary produced by the DDL compiler is a set of files that form a permanent record of your data base organization. The dictionary provides ENFORM with information about each record in the data base. If you write an ENFORM query, any records specified in the query must be described with a DDL RECORD statement and the RECORD statement must be compiled into a data dictionary.

Use the ?SHOW Command to Examine a Record Description. If you have forgotten the name of a field or its data category, use the ?SHOW command from within the ENFORM process to examine the dictionary record description. First, issue the ENFORM OPEN statement which causes ENFORM to read a copy of the record description into its internal table. Next issue the ?SHOW record command which causes ENFORM to display the record description. For example:

```
OPEN parts;
?SHOW parts
```

causes ENFORM to display:

```
01 A 0:37             PARTS: $MKT.SAMPLE.PARTS.
 02 N 0:4   ,P-KEY    PARTNUM.
 02 A 4:18  ,A-KEY    PARTNAME.
 02 N 22:4            INVENTORY.
 02 A 26:3            LOCATION.
 02 N 29:8            PRICE.
```

Use ENFORM to Create Dictionary Reports. Use ENFORM to produce reports about the dictionary that provide:

- Data base documentation for the individuals using the data base.
- Data base analysis information that can be helpful when modifications or additions to the dictionary are considered.

To produce dictionary reports, use an ENFORM source file that resides in the file $SYSTEM.SYSTEM.DDQUERYS. The ENFORM source file can be modified to produce reports tailored to answer specific questions. The ENFORM source file consists of twelve queries that produce twelve different reports. Each query is a separate section of the source file; therefore, the queries can be run either individually or in combination.

To obtain the reports, a dictionary describing the file structure of the seven DDL dictionary files must reside on $SYSTEM.DDL. Entering the following lines produces all twelve reports:

```
:VOLUME $yourvol.yoursubv      !establish the default volume
:ENFORM/IN $system.system.ddquerys,OUT $s/
```

Examples of these reports can be found in the *Data Definition Language (DDL) Reference Manual.*

USING COBOL, FORTRAN, AND TAL DATA DECLARATION SOURCE CODE. DDL optionally produces data declaration source code for the COBOL, FORTRAN, and TAL programming languages. Including this source code in an application program used to maintain or alter the data base reduces programming effort and enforces consistency of data handling.

Creating Data Base Files

Use the File Utility Program (FUP) to create the actual physical files that store the data for your data base. If you use a FUP file creation source file produced by DDL, consider editing the file before using it. Edit the file to:

- Increase the block size. Unless you have specified the DDL FUPBLOCKSIZE command, the default block size DDL writes to the FUP source file is 512 bytes. This block size might be adequate for lightly used files; however, providing a larger block size avoids a level of indexing that might slow processing for heavily used large files.

- Adding a SET EXTENT command to increase the extent sizes. The FUP default extent sizes of one page for the primary extent and one page for the secondary extent might not be large enough.

Refer to the *GUARDIAN Operating System Command Language and Utilities Manual* for more information about FUP.

Loading Data Base Files

Choose a method of loading the data into the physical files. Use either an application program or the ENABLE subsystem.

Using an application program is most advantageous when the data already exists in some machine readable form (such as on a disc file or a tape), but some data conversion operation must be performed before the data is loaded. When an application program is used, include the data declaration source code optionally produced by DDL to insure that the data is handled in a consistent manner.

Using the ENABLE subsystem is most advantageous when the data is to be entered from a terminal. ENABLE accesses the dictionary record description of the data being loaded, generates an interactive application under PATHWAY, and provides a screen for terminal access to a single record type. The terminal display screen consists of field names and values in columnar format and contains appropriate annotations to assist the operator in entering data. For more information about ENABLE, refer to the *ENABLE Users Guide.*

Occasionally FUP is used to either load or reload data. For example, FUP could be used if the data already exists in a machine readable form and data conversion is not needed. If FUP is used, remember to load alternate key files. If an alternate key file exists, but is empty, ENFORM retrieves no data when that alternate key path is chosen as its search strategy.

SECTION 3

DEVELOPING AN ENFORM QUERY

The primary aim of entering a query is to retrieve the information you want from the data base. Depending on your needs, ENFORM provides the information in one of the following forms: as a report, as a new physical file, or as records transmitted to a host language program. This section discusses the steps needed to produce a report or a new physical file. (Refer to Section 6 for information about transmitting records to a host language program). The steps are:

1. Establishing the environment to be used for the query by using statements and commands such as ?DICTIONARY, OPEN, ?ASSIGN, DECLARE, and SET.

2. Establishing relationships between record descriptions by using the LINK statement or the WHERE clause.

3. Specifying the information selected by using the LIST or FIND statement.

4. Restricting the information selected by using the WHERE clause.

5. Specifying the sorting and grouping of selected information by using the BY, BY DESC, ASCD, or DESC clauses.

6. Specifying computations for selected information by using the SUBTOTAL, TOTAL, PCT, or CUM clauses.

7. Formatting a report by using clauses and statements such as TITLE, HEADING, AT END, AS, SPACE, or TAB.

The first five steps apply to any ENFORM query that produces a report or creates a new physcial file. The last two steps apply only to a query that produces a report. The following paragraphs discuss these steps and offer a brief overview of the various operations you can perform during an ENFORM session (the period of time that begins when you enter the ENFORM command and ends when you exit the ENFORM subsystem).

ESTABLISHING THE QUERY ENVIRONMENT

Establishing the query environment is the first step in developing a query. Table 3-1 shows the ENFORM statements and commands that establish the query environment. If your dictionary resides on your current volume and subvolume, the OPEN statement is the only statement or command that is required. If your dictionary resides on a different volume or subvolume, either the DICTIONARY statement or the ?DICTIONARY command is also required.

Table 3-1. Statements and Commands Used to Establish the Query Environment

Statement	Command	Function
DICTIONARY	?DICTIONARY	Identifies the volume and subvolume on which the dictionary resides.
OPEN		Identifies the record descriptions used in the query.
	?ASSIGN	Assigns a record description to a different physical file.
DECLARE		Defines user elements.
SET		Initializes user elements and resets operational variables.

Identifying the Dictionary

To identify the volume and subvolume on which your dictionary resides, use one of the following:

- The dictionary parameter in the ENFORM command.
- The DICTIONARY statement.
- The ?DICTIONARY command.

If the dictionary is not identified, ENFORM assumes it resides on your current volume and subvolume.

If the dictionary is identified in the ENFORM command and you want to change dictionaries, use either the DICTIONARY statement or the ?DICTIONARY command. For example, the following DICTIONARY statement indicates to ENFORM that you want change dictionaries from the one residing on *$mkt.sample* to the one residing on *$data.test*:

```
:ENFORM $mkt.sample
  ...
>DICTIONARY $data.test;
```

Identifying Record Descriptions

Use the OPEN statement to identify the record descriptions used in your query. When the OPEN statement is issued, the query compiler obtains the record description of the identified record from the dictionary and stores this information in its internal table. Figure 3-1 shows the effect of an OPEN statement that identifies the *parts* record description.

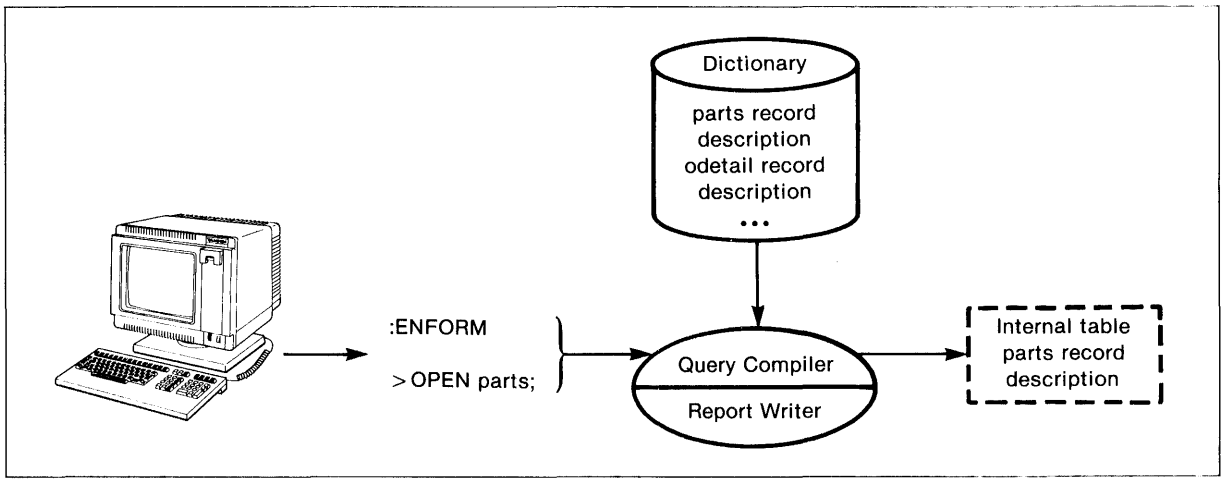


Figure 3-1. Effect of an OPEN statement

Note that the OPEN statement does not actually open the physical file in which the *parts* record occurrences are stored.

Use OPEN AS COPY OF when a record description is designed so that record occurrences stored in a physical file relate to other record occurrences stored in the same physical file. OPEN AS COPY OF allows you to use either a LINK statement or a linking WHERE clause (both are described later in this section) to establish a linking relationship between the record occurrences. Establishing such a linking relationship is impossible without using OPEN AS COPY OF because ENFORM requires that a LINK statement or a linking WHERE clause specify two different record names.

Consider the record description shown in Figure 3-2.

```
RECORD employ.
        FILE IS    $mkt.sample.employ          KEY SEQUENCED.
   02 empl-no              PIC  9(4) .
   02 emp-name             PIC  X(18).
   02 dept.
      05 reg-num            PIC  99.
      05 branch-num         PIC  99.
   02 salary               PIC  999999.
   02 mgr-id               PIC  9(4).
   KEY IS empl-no.
   KEY "en" IS empl-name.
   KEY "dp" IS dept.
END
```

Figure 3-2. Record Description For OPEN AS COPY OF

The following OPEN AS COPY OF statement logically makes a duplicate copy of *employ* and names it *empdup*. Notice that the OPEN statement for the record description being duplicated must precede the OPEN AS COPY OF statement.

```
OPEN employ;
OPEN empdup AS COPY OF employ;
LINK employ.mgr-id TO empdup.empl-no;
```

Specification of the OPEN AS COPY OF and LINK statements allows the names of all employees and their managers to be listed.

Assigning Record Descriptions To Different Physical Files

Use the ?ASSIGN command when the record occurrences you want to retrieve are not stored in the physical file specified in the DDL FILE IS clause. (Refer to the discussion of the Data Definition Language (DDL) in Section 2 for more information about the FILE IS clause.) When you specify the ?ASSIGN command, the query compiler stores the new physical file name in the internal table and passes this information to the query processor when you issue either a LIST or FIND statement. Remember, an OPEN statement for the record description must also be issued.

Figure 3-3 shows the effect of an ?ASSIGN command.

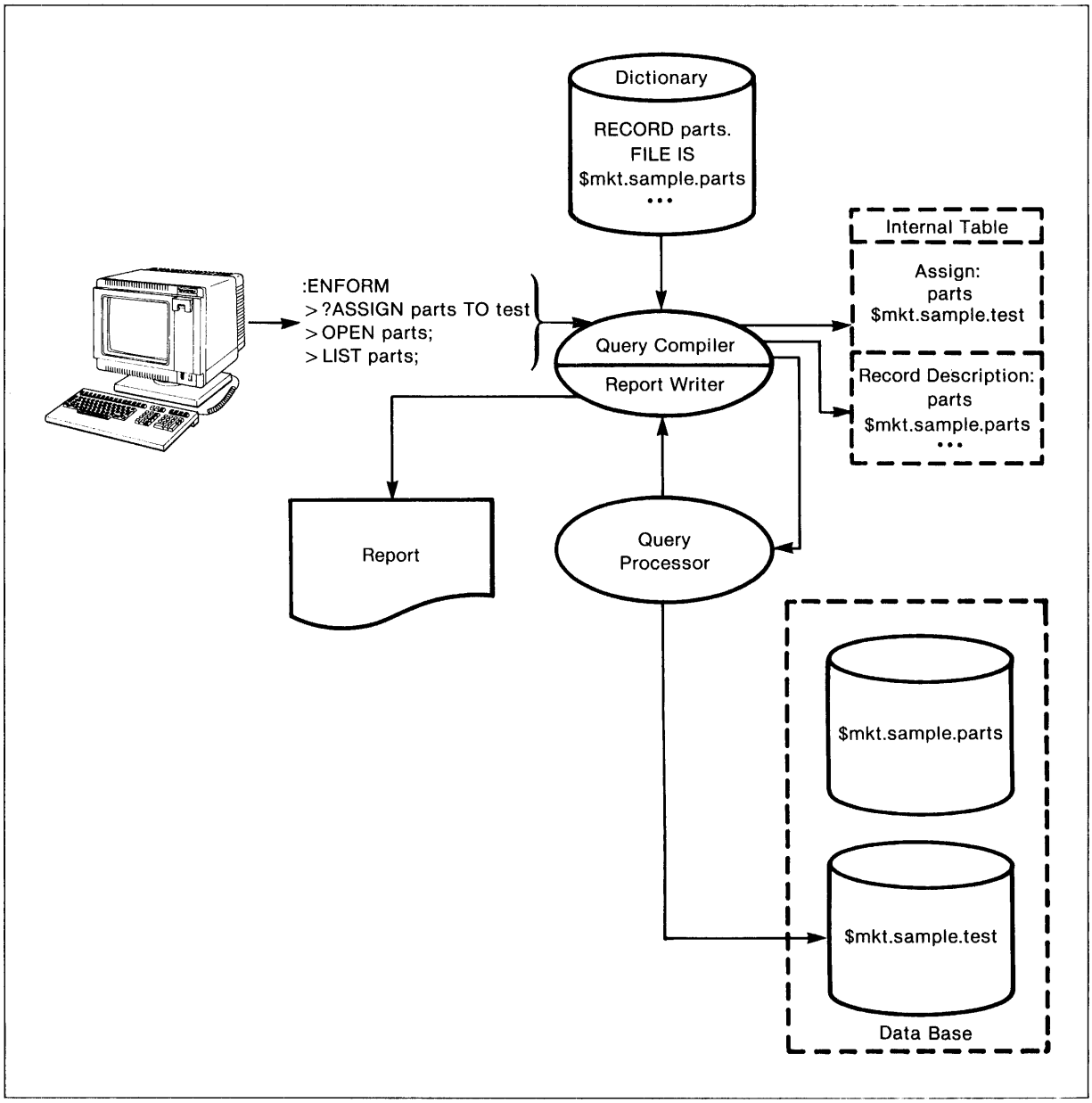


Figure 3-3. The Effect of an ?ASSIGN Command

Using the ?ASSIGN command is useful when you want to test a query that would normally retrieve a large amount of data. Instead, copy a small portion of the record occurrences to be retrieved to a new physical file, specify that file name in an ?ASSIGN command, and then test the query.

Using the ?ASSIGN command also makes maintenance easier. If the ?ASSIGN command is specified and the data file is moved or renamed, simply change the ?ASSIGN statement instead of recompiling the dictionary.

Defining User Elements

Use the DECLARE statement to define any user elements to be used in the query. The DECLARE statement defines user variables, user tables, and user aggregates. The DECLARE statement also optionally establishes the internal format, the default display format, and the default heading of user variables or tables.

When the DECLARE statement is issued, the query compiler stores the information specified in the internal table. This information remains in the internal table until a CLOSE statement for the user element is issued, a ?DICTIONARY command or DICTIONARY statement is issued, or the ENFORM session terminates.

Issue another DECLARE statement specifying the same user variable or table name to change the internal format, the default display format, or the default heading. If you issue another DECLARE statement that specifies the name of a previously defined user aggregate, ENFORM issues an error message.

Use the SET statement to provide an initial value for a user variable or user table.

Setting Option Variables

The option variables store values that ENFORM uses when performing certain operations. Each option variable has a default value that you can reset by using the SET statement. When the value of an option variable is reset, the new value remains in effect for the duration of the current ENFORM session. By using the SET statement to reset the default value of an option variable, you can:

- Establish a new report format
- Affect the processing strategy that ENFORM uses during the current session
- Control the size of the target file that ENFORM produces when processing a LIST or FIND statement
- Generate statistics that you can use to improve processing efficiency
- Control the amount and kind of information that ENFORM displays at your terminal.

For example, if you have a terminal that does not have a scrolling mechanism, you might want to control the number of lines of information that ENFORM displays on your terminal at any one time. To do this, set the value of the @DISPLAY-COUNT option variable to 24:

```
SET @DISPLAY-COUNT TO 24;
```

When you set @DISPLAY-COUNT to 24, ENFORM displays a maximum of 24 lines of information. To indicate that you are ready for the next 24 lines of information, press the terminal RETURN key.

As another example, consider resetting the left-hand margin that ENFORM uses when producing a report. To do this, set the value of the @MARGIN option variable to the column in which you want the report to begin printing. If you want ENFORM to begin printing in column 5, set @MARGIN to 5:

```
SET @MARGIN TO 5;
```

Refer to Section 5 for other examples that demonstrate the advantages of resetting the option variables.

CONNECTING RECORD DESCRIPTIONS TO FORM NEW RELATIONSHIPS

ENFORM allows you to connect two or more record descriptions in a relationship that establishes a new logical record description. The process of establishing a connecting relationship is called linking. During the linking process, ENFORM links the dictionary record descriptions of the records and builds logical record occurrences composed of record occurrences from both record descriptions.

The number of target records (the data base information retrieved by the query processor) resulting from a query where record descriptions are linked depends on the data values in the linking fields. The number of target records ranges from a minimum value of zero to a maximum value that is the product of the number of record occurrences associated with each record description.

The minimum number of resulting target records occurs when no data value of the linking field in the first record description is found in the linking field of the second record description.

The maximum number of resulting target records occurs when the linking field has the same value in every record occurrence for both record descriptions. In some cases, specifying a link that returns the maximum number of target records causes unexpected results. For example, suppose that 1000 record occurrences are associated with *a-rec* and 3000 record occurrences are associated with *b-rec*. Suppose also that the linking field (*link-field*) in all the record occurrences contains the value SAME STRING. If the following query is issued:

```
OPEN a-rec,b-rec;
LINK a-rec TO b-rec VIA link-field;
LIST a-rec,b-rec;
```

The number of resulting target records is 3 million, the product of the number of record occurrences associated with *a-rec* and the number of record occurrences associated with *b-rec*. If your query returns an unexpectedly large number of target records, examine the links specified.

Use the following guidelines for links:

- Link fields that belong to the same data category (either both numeric or both alphanumeric). Fields that belong to different data categories cannot be linked.
- Link on a full key or the left portion of a composite key; if the right portion of a composite key (described in Section 2) is used, be sure it is necessary. Such a link requires more processing time and might cause unexpected results.
- Link fields that contain the same field values. Linking fields that do not contain the same field values results in increased processing time and possibly unwanted results.
- Define only the links that your query needs. Defining more links than your query needs can unnecessarily restrict the information returned. If necessary, clear unwanted links.
- Use efficient links. The most efficient links are usually between fields that are primary keys, alternate keys, or fields that are described with a SEQUENCE IS clause in the DDL record description.

The ENFORM statements, clauses, and commands used to connect, clear, and display links are shown in Table 3-2.

Table 3-2. Establishing and Clearing Relationships

Statement	Clause	Command	Function
LINK LINK OPTIONAL			Creates a session-wide link; used to link record descriptions for a query.
	WHERE		Creates a temporary link for the associated LIST or FIND statement.
		?SHOW LINK	Displays session-wide links currently in effect.
DICTIONARY DELINK DELINK OPTIONAL CLOSE		?DICTIONARY	Clears links from the internal table.

Making Session-Wide Links

ENFORM provides you with two statements that make session-wide links. These statements are the LINK statement and the LINK OPTIONAL statement. While both statements establish links, the link established by a LINK statement differs from the link established by a LINK OPTIONAL statement. The following paragraphs discuss both statements and describe their differences.

USING THE LINK STATEMENT. When you enter the LINK statement, ENFORM treats the linked record descriptions as one logical record description. ENFORM builds record occurrences for the new logical record description by selecting matching records from the data files associated with the linked record descriptions. Records match when both contain the same data value in their linking fields. For example, consider the following:

```
OPEN employee,region;
LINK region.manager TO employee.empnum;
```

Figure 3-4 shows the logical record occurrences created by the preceding LINK statement. To build the logical record occurrences, ENFORM obtains the value of the *manager* field from the first *region* record. ENFORM then searches the *employee* records looking for a matching data value in the *empnum* field. If ENFORM finds a matching value, it builds a logical record occurrence for the matching *region* and *employee* record occurrences.

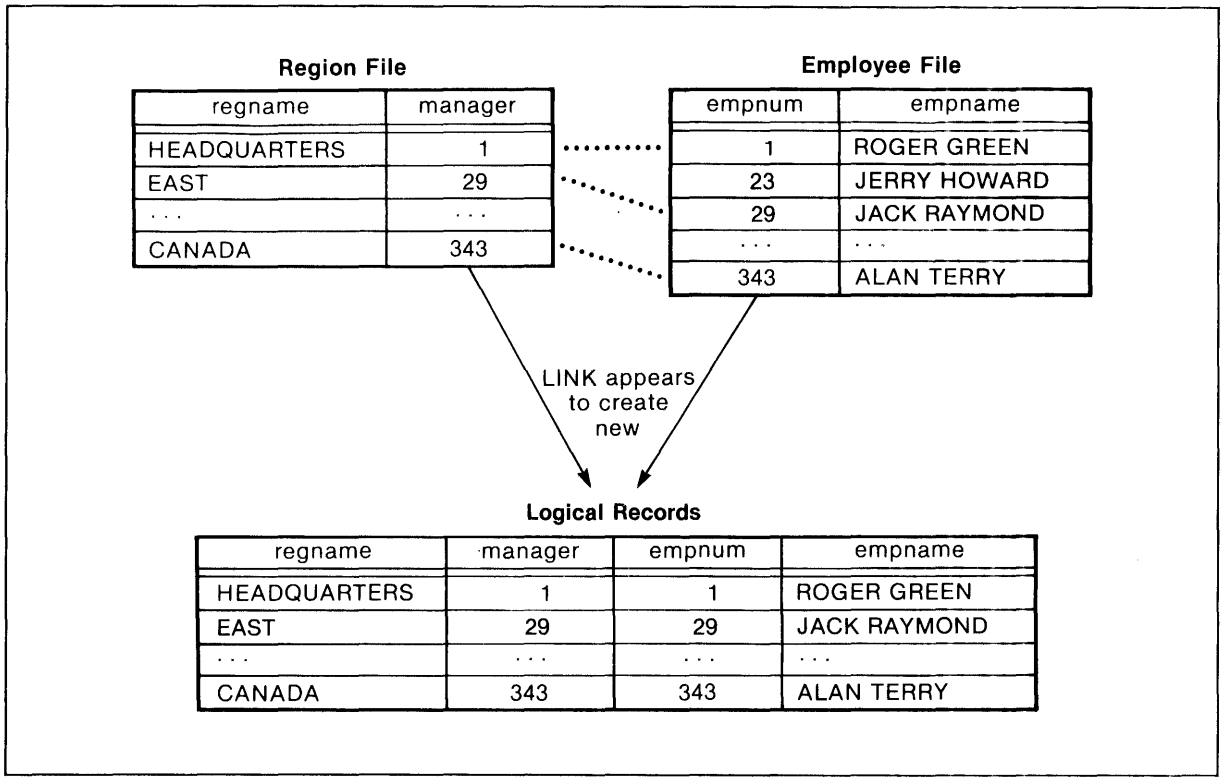


Figure 3-4. The Process of Finding Matching Values

In Figure 3-4, each data value in the *manager* field matches a data value in the *empnum* field. Figure 3-5 shows a diagram of the new logical record occurrences built when a matching value (*empnum* = 29) is missing and the same LINK statement is issued.

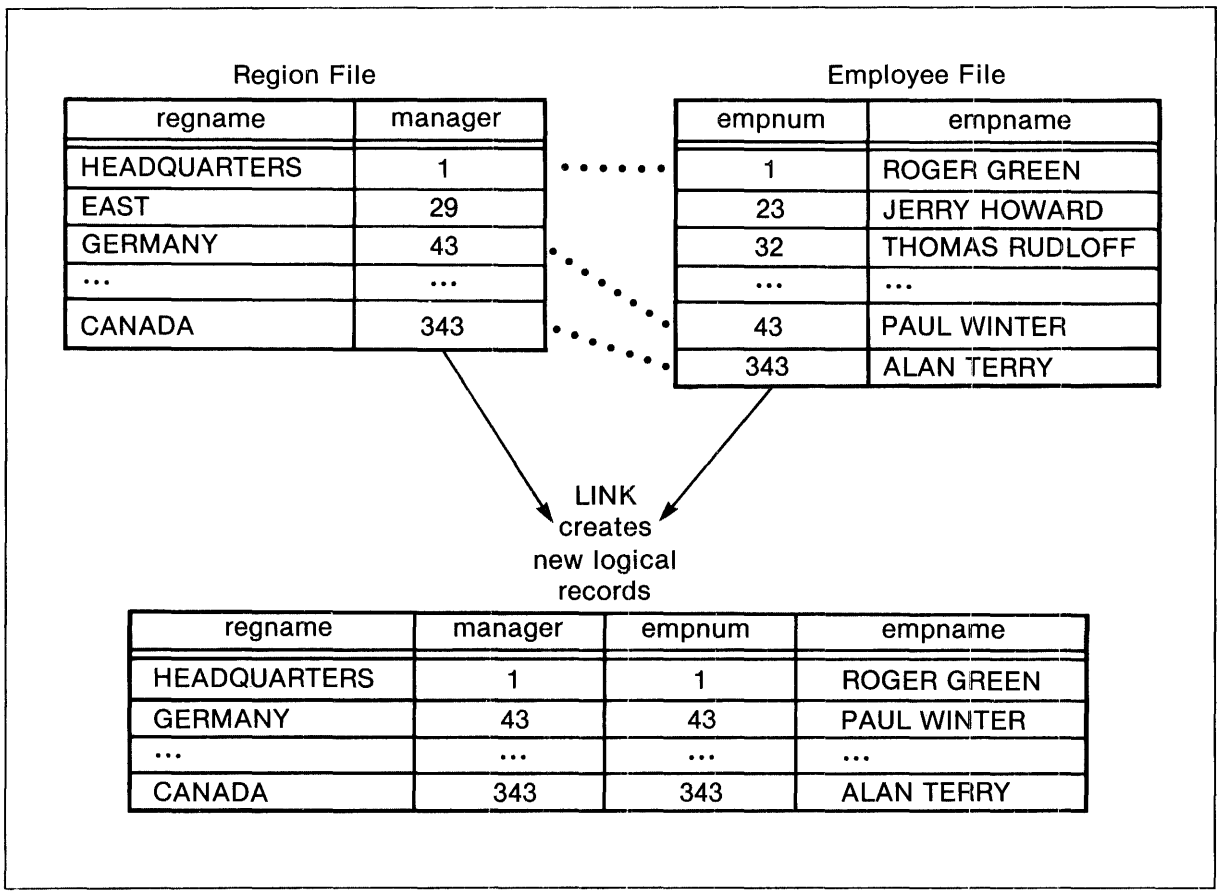


Figure 3-5. Logical Records Built When a Matching Value is Missing

USING THE LINK OPTIONAL STATEMENT. The major difference between a link established by a LINK statement and a link established by a LINK OPTIONAL statement is the way that ENFORM builds the logical record occurrences for the new logical record description. When you specify the LINK statement, ENFORM builds the logical record occurrences by selecting matching records from the data files associated with the linked record descriptions. When you specify the LINK OPTIONAL statement, ENFORM builds a set of logical records as follows:

- ENFORM builds one logical record occurrence for each set of matching record occurrences. Each of these logical record occurrences contains data values from both matching record occurrences.

- If any record occurrence associated with the record description on the left side of the LINK OPTIONAL statement does not appear in the set of logical record occurrences, ENFORM builds a logical record occurrence for that record. In each of these logical record occurrences, ENFORM supplies null values (blanks) for the fields that correspond to the record description specified on the right side of the LINK OPTIONAL statement.

For example, consider the following:

```
OPEN region,employee;
LINK region.manager TO OPTIONAL employee.empnum;
```

Figure 3-6 shows the record occurrences associated with both the *region* and the *employee* record descriptions. This figure also shows the logical record occurrences built for the LINK OPTIONAL statement. Notice that some of the logical record occurrences contain blanks for the fields from the employee records. These fields are blank because no employee record matches the particular region record. For example, ENFORM includes the region record occurrence where *manager* has a value of 29 even though there is no matching record occurrence in *employee*.

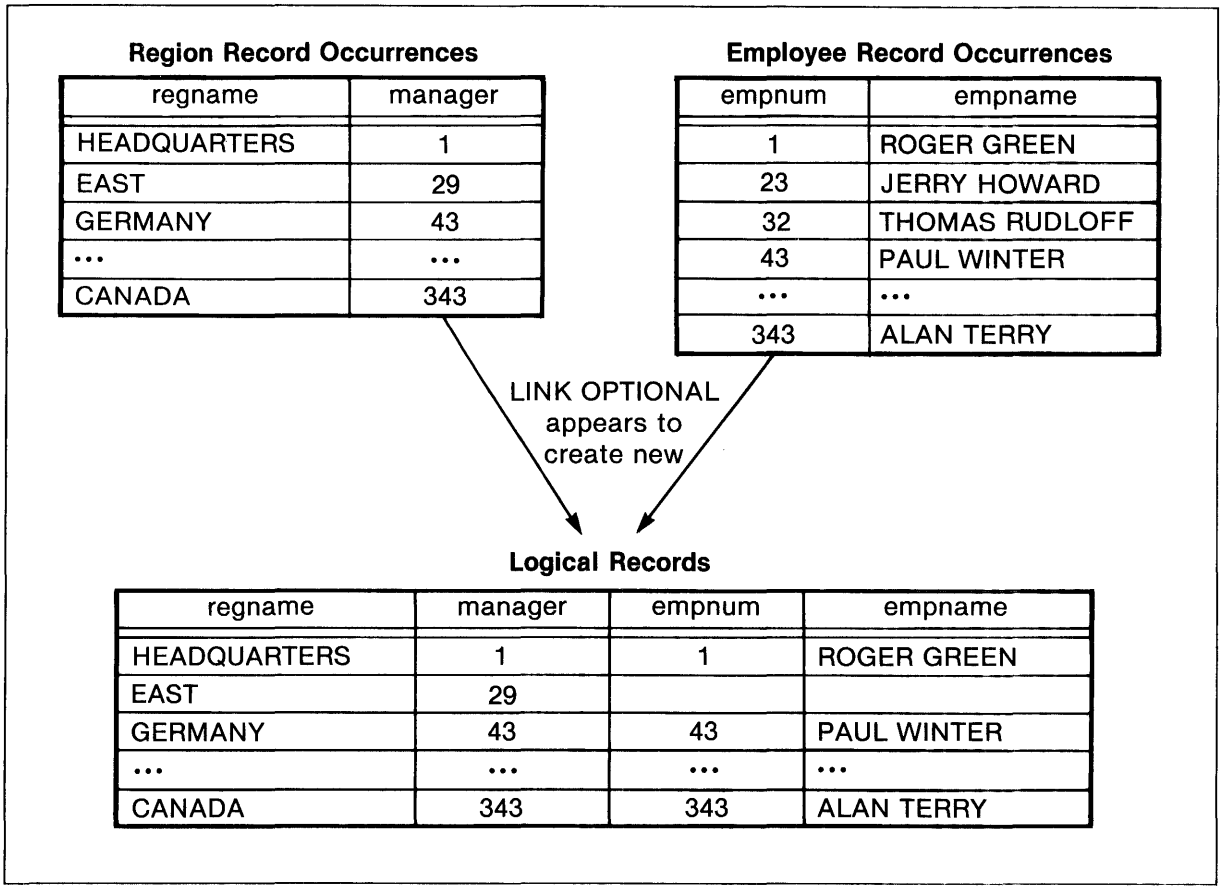


Figure 3-6. Diagram of LINK OPTIONAL Where Region is Linked to Employee

If the order of the record descriptions in the preceding LINK OPTIONAL statement is reversed:

```
LINK employee.empnum TO OPTIONAL region.manager;
```

the resulting logical record occurrences are very different. As Figure 3-7 shows, ENFORM includes all the *employee* records in the logical record occurrences even though a matching *region* record does not exist; however, ENFORM does not include any *region* record that does not match an *employee* record.

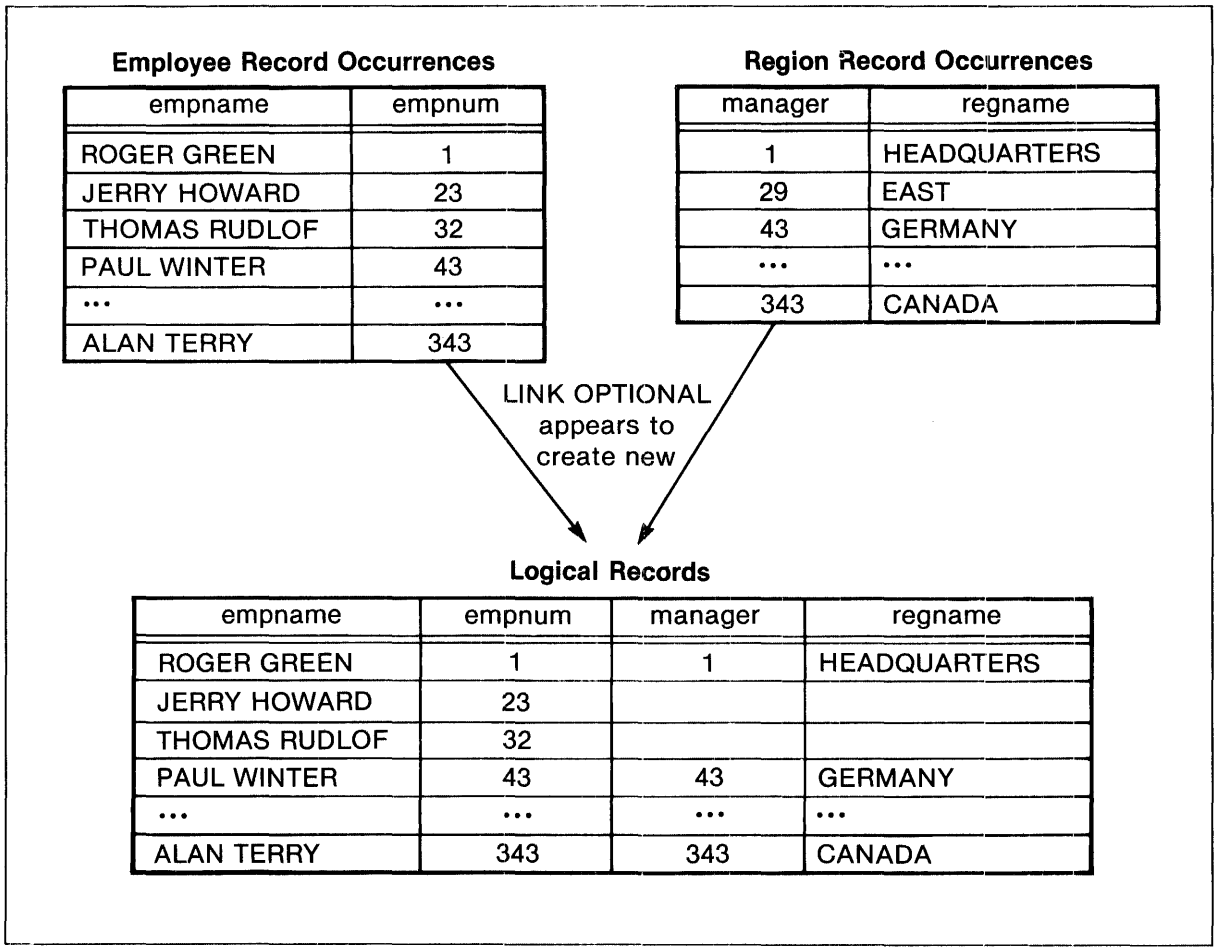


Figure 3-7. Diagram of LINK OPTIONAL Where Employee is Linked to Region

SESSION-WIDE LINKS AND THE WHERE CLAUSE. When your query specifications contain a WHERE clause, ENFORM uses any request qualifications specified in the WHERE clause to produce the target records (the records from which your ENFORM output is produced). ENFORM takes the logical expressions specified in your WHERE clause and converts them into conjunctive normal form. This means that ENFORM converts the logical expressions into one or more "terms". These "terms" consist of logical expressions that are connected to other logical expressions by the boolean operator AND. Each "term" can have "subterms" that are connected by the boolean operator OR. Note that what appears as a "term" in your WHERE clause might not be the same as a "term" in the converted WHERE clause. Refer to Appendix D of the *ENFORM Reference Manual* for more information about the "terms" of a WHERE clause.

Normally, ENFORM produces target records by evaluating all of the "terms" in the WHERE clause and selecting only those logical record occurrences that satisfy all the "terms". If your query specifications contain both a WHERE clause and LINK OPTIONAL statements, ENFORM might not use every "term" in a converted WHERE clause to evaluate every given logical record occurrence. Before ENFORM uses a "term" to evaluate a given logical record occurrence, ENFORM examines both the "term" and the logical record occurrence. ENFORM determines whether the "term" references a record description that is "non-contributing" for the given logical record occurrence. For a given logical record occurrence, a record description is "non-contributing" if the logical record occurrence does not contain any data values from the data file associated with the record description. (Refer to the *ENFORM Reference Manual* for more information about "non-contributing" record descriptions.) ENFORM evaluates logical record occurrences by using the "terms" in the converted WHERE clause as follows:

- If all of the record descriptions referenced in a "term" contribute to a given logical record occurrence, ENFORM uses the "term" to evaluate the logical record occurrence.

- If some of the record descriptions referenced in a "term" are "non-contributing" for a given logical record occurrence, ENFORM does not use the "term" to evaluate the logical record occurrence.

For example, suppose you issue the following LINK OPTIONAL statement:

```
LINK employee TO OPTIONAL region VIA regnum;
```

If you then enter the following query:

```
LIST empname, regname, WHERE empnum < 100 AND
                          regname = "GERMANY";
```

ENFORM builds the logical record occurrences from the record occurrences shown in Figure 3-8. ENFORM then produces the target records also shown in that figure.

Employee Record Occurrences

empnum	empname	regnum
1	ROGER GREEN	99
23	JERRY HOWARD	2
32	THOMAS RUDLOFF	5
43	PAUL WINTER	5
49	MARK MONTEMORRA	7
65	SUSAN HENDERSON	6
104	DAVID STRAND	2
...	...	...
343	ALAN TERRY	4

Region Record Occurrences

regnum	regname
1	EAST
2	CENTRAL
3	WEST
4	CANADA
5	GERMANY
6	ENGLAND
99	HEADQUARTERS

Logical Record Occurrences Built for the Link

empnum	empname	regnum	regnum	regname
1	ROGER GREEN	99	99	HEADQUARTERS
23	JERRY HOWARD	2	2	CENTRAL
32	THOMAS RUDLOFF	5	5	GERMANY
43	PAUL WINTER	5	5	GERMANY
49	MARK MONTEMORRA	7		
65	SUSAN HENDERSON	6	6	ENGLAND
104	DAVID STRAND	2	2	CENTRAL
...	...	...	...	...
343	ALAN TERRY	4	4	CANADA

Target Records

empname	regname
THOMAS RUDLOFF	GERMANY
PAUL WINTER	GERMANY
MARK MONTEMORRA	

Figure 3-8. Report produced when both LINK OPTIONAL and WHERE Clause Specified

ENFORM converts the preceding WHERE clause into two "terms":

```
empnum < 100 AND      ------------> > first term

regname = "GERMANY"   ------------> > second term
```

Notice that ENFORM produces a target record from the logical record occurrence whose *empnum* value is 49. ENFORM does not use the second "term" (regname = "GERMANY") to evalute this logical record occurrence because the *region* record description is "non-contributing" for this logical record occurrence. (This record description is "non-contributing" because the *regnum* value (7) in the contributing *employee* record occurrence does not match a *regnum* value in the *region* record occurrences.)

CLEARING UNNECESSARY SESSION-WIDE LINKS. You can clear unnecessary session-wide links (links created by either the LINK statement or the LINK OPTIONAL statement) by using one of the following:

- A CLOSE statement which clears all links referencing the record descriptions being closed.
- A DELINK statement which clears the specified link.
- Either a DICTIONARY statement or a ?DICTIONARY command that clears the entire internal table.

Unnecessary links take up memory space, produce undesirable results in subsequent queries, and might lengthen processing time. After clearing unnecessary links, you can use the ?SHOW link command to verify the removal of the links.

EXAMINING SESSION-WIDE LINKS. You use the ?SHOW link command to examine session-wide links. For example, if you enter:

```
LINK parts TO fromsup VIA partnum;
LINK employee TO OPTIONAL region VIA regnum;
```

and later enter a ?SHOW LINK command, ENFORM displays:

```
PARTS.PARTNUM is linked to FROMSUP.PARTNUM
EMPLOYEE.REGNUM is linked optional to REGION.REGNUM
```

Establishing Links For the Current Query

You can establish a link that applies only to the current query by including a WHERE clause in either a LIST or FIND statement. If you want to establish a link with a WHERE clause, at least one of the terms of the WHERE clause must reference two record descriptions. (Refer to the *ENFORM Reference Manual* for an explanation of the terms of a WHERE clause). When you establish such a link, the link applies only to the query of which it is a part. For example, in the following query the WHERE clause links *parts* to *fromsup*:

```
OPEN parts, fromsup;
LIST parts.partnum,
     partname,
     suppnum,
  WHERE parts.partnum EQ fromsup.partnum;
```

Report:

```
 Part
Number       PARTNAME        SUPPNUM
------  ------------------   -------

   212  SYSTEM 192KB CORE          1
   244  SYSTEM 192KB SEMI          1
  1403  PROC 96KB SEMI             1
  ...   ...                      ...
  6603  TERM HARD COPY             2
  7102  CABINET LARGE             10
  7301  POWER MODULE               1
```

If you issue another query later in the same session, the link established by the WHERE clause is no longer in effect. Suppose, for example, after entering the preceding query, you enter:

```
LIST parts.partnum, partname, fromsup.suppnum;
```

Since the link established by the WHERE clause in the preceding query no longer exists, ENFORM displays an error message stating that at least one record has no link relating it to any other record.

You can use a WHERE clause to link more than one record description. For example, in the following query, the WHERE clause temporarily links *parts, fromsup,* and *supplier:*

```
OPEN parts, fromsup, supplier;
LIST parts.partnum, partname, fromsup.suppnum, suppname,
  WHERE parts.partnum EQ fromsup.partnum
    AND fromsup.suppnum EQ supplier.suppnum;
```

Report:

```
 Part
Number       PARTNAME         SUPPNUM        SUPPNAME
------  ------------------   -------  ------------------

   212  SYSTEM 192KB CORE          1  TANDEM COMPUTERS
   244  SYSTEM 192KB SEMI          1  TANDEM COMPUTERS
  1403  PROC 96KB SEMI             1  TANDEM COMPUTERS
  ...   ...                      ...  ...
  7102  CABINET LARGE             10  STEELWORK INC
  7301  POWER MODULE               1  TANDEM COMPUTERS
```

Using a WHERE clause to link record descriptions with conditional operators other than EQUAL is also possible.

Combining Links

ENFORM allows you to combine links initiated by LINK statements with links initiated by LINK OPTIONAL statements or links initiated by a WHERE clause. If you include LINK OPTIONAL statements in your query specifications, however, you must follow the rules described in the *ENFORM Reference Manual.*

The following paragraphs show examples of queries for which multiple links are necessary. Many of these examples also show sketches of the links, where:

A sketch of the links that affect a query can often help you find:

- "silly" or unnecessary links. If, for example, your query returns unexpected results, you can use the ?SHOW LINK command to determine which session-wide links affect your query. If you then sketch all the links (including any links initiated by a WHERE clause), you might discover the reason that you have obtained the unexpected result. (For example, you might have specified a link that unnecessarily restricts the target records returned for the query.)
- a link that violates one of the rules specified for the LINK OPTIONAL statement. If ENFORM returns an error message that indicates you have specified an illegal link, a sketch of the links often helps you find the illegal link.

You might specify a combination of links if you want to obtain information from the data files associated with more than two record descriptions. For example, suppose that you want to produce a report that lists the orders taken by your company, the price that your company can charge for each ordered part, and the price that your company must pay for each ordered part. To produce such a report, ENFORM must obtain information from five record descriptions: *order*, *parts*, *odetail*, *fromsup*, and *supplier*. As the following query specifications show, you could initiate these links with LINK statements and a WHERE clause:

```
OPEN order, parts, odetail, fromsup, supplier;
LINK order TO odetail VIA ordernum;
LINK odetail TO parts VIA partnum;
LINK parts TO fromsup VIA partnum;
LIST BY order.ordernum HEADING "No.",
     BY partname HEADING "Part",
        SUM ((quantity * price) OVER partname) HEADING "Our/Price",
        suppname HEADING "Supplier",
        (quantity * partcost) HEADING "Our/Cost"
  WHERE fromsup.suppnum = supplier.suppnum;
```

A sketch of the links in the preceding query appears as follows:

```
order <---> odetail <---> parts <---> fromsup <---> supplier
```

Because you have linked the record descriptions, ENFORM can build a set of logical record occurrences from which it produces the target records for the following report (note: this report has been edited so that it will fit on the manual page):

```
                              Our                                  Our
No.          Part           Price          Supplier              Cost
---  ------------------  -----------  ------------------  ---------

 21  DECIMAL ARITH          3000.00  TANDEM COMPUTERS      2700.00
     DISC 160MB           147000.00  INFORMATION STORAGE  40000.00
                                     MAGNETICS CORP       38600.00
                                     DATADRIVE            39000.00
     MEM MOD 96K MOS        19200.00  TANDEM COMPUTERS     18900.00
     SYSTEM 192KB SEMI      87000.00  TANDEM COMPUTERS     85000.00
 25  ASYNC CONTROLLER        5800.00  TANDEM COMPUTERS      5500.00
     MAG TAPE DR 8/16       16000.00  MAGNETICS CORP        6200.00
                                     DATADRIVE             6250.00
     SYSTEM 192KB SEMI      87000.00  TANDEM COMPUTERS     83000.00
     TERM CRT PAGE          30000.00  DATA TERMINAL        11000.00
                                     DISPLAY INC          26000.00
 ..  ...                    ...       ...                  ...
```

If you need to see all of the data stored for a particular record description, you can specify that record description on the left side of a LINK OPTIONAL statement. For example, suppose that you want a report that lists all of the orders for your company, all the employees who took the orders, and the region to which those employees belong. The specifications for this query might appear as follows:

```
OPEN order, employee, region;
LINK order.salesman TO OPTIONAL employee.empnum;
LINK employee TO OPTIONAL region VIA regnum;
LIST ordernum, empname, regname;
```

A sketch of the links in the preceding query appears as follows:

The report produced by this query is:

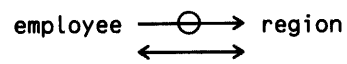
```
ORDERNUM       EMPNAME          REGNAME
--------  ------------------  ------------

      21  GEORGE FORSTER      EAST
      25  JONATHAN MITCHEL    WEST
      30  MARTIN SCHAEFER     GERMANY
      32  TOM HALL            EAST
     ...  ...                 ...
     122  OTTO SCHNABL        GERMANY
     149
```

Notice that the report entry for order number 149 contains neither a salesman name nor a region name. This type of entry often provides a clue that the data stored within your data base is inconsistent. For example, the entry in the preceding report could be caused by a data entry error in either the data file associated with the *order* record description or the data file associated with the *employee* record description.

To check the consistency of data within your data base, you can produce an exception report. Such a report discovers inconsistencies between the data stored for two or more record descriptions. For example, suppose that you want to check the validity of the region numbers entered in the *employee* record occurrences. Since you are verifying the region numbers in *employee*, you must link *employee* to *region* with a LINK OPTIONAL statement. By specifying a WHERE clause, you can request a report that lists only those employees whose region numbers are invalid. The specifications for this query might appear as follows:

```
OPEN employee, region;
LINK employee TO OPTIONAL region VIA regnum;
LIST empname, employee.regnum,
   WHERE employee.regnum <> region.regnum;
```

A sketch of the links in the preceding query appears as follows:



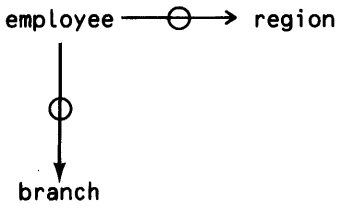
Since *employee* is optionally linked to *region*, the logical record occurrences built for the linked record descriptions contain all of the *employee* records. The WHERE clause tells ENFORM that you are interested only in those logical record occurrences that do not have data values in the fields that correspond to the *region* record descriptions. (Note that the link established by the WHERE clause is not illegal because it references the same record descriptions as those referenced in the LINK OPTIONAL statement.) ENFORM, therefore, produces the following report:

```
     EMPNAME          REGNUM
-----------------  ------

ALDEN SPROWLES          67
JOAN ZIMMERMAN          11
BOB CHAPIN              17
LIZ CHAMBERS            59
```

If you want to produce an exception report that lists all the employees with invalid region numbers and invalid branch numbers, you could use LINK OPTIONAL statements to link *employee* to both *region* and *branch*. In this case, you could use a SUPPRESS clause to specify that the report is to contain only the names of those employees who have invalid region or branch numbers. (You cannot use a WHERE clause because such a clause would establish a link that violates the rules for the LINK OPTIONAL statement.) The specifications for this query might appear as follows:

```
OPEN employee, branch, region;
LINK employee TO OPTIONAL region VIA regnum;
LINK employee.dept TO OPTIONAL branch.primkey;
LIST empname, employee.regnum, employee.branchnum
  SUPPRESS WHERE employee.regnum = region.regnum
    AND employee.dept = branch.primkey;
```

A sketch of these links appears as follows:



Since *employee* is linked optionally to both *region* and *branch*, the logical record occurrences built for the linked record descriptions contain all of the *employee* records. The SUPPRESS WHERE clause tells ENFORM that you are interested only in those logical record occurrences that do not have data values in the fields that correspond to the *region* and *branch* records. ENFORM, therefore, produces the following report:

EMPNAME	REGNUM	BRANCHNUM
ALDEN SPROWLES	67	2
JOAN ZIMMERMAN	11	3
BOB CHAPIN	17	99
CHARLES WONG	9	0
LIZ CHAMBERS	59	1

By specifying multiple links, you can use ENFORM to produce a report that lists an item and its component items. Suppose, for example, that your data base contains the files described in the following record descriptions:

```
RECORD newpart.
FILE IS newpart  KEY-SEQUENCED.
02 p-no          PIC 9(4).
02 p-name        PIC X(20).
02 p-price       PIC 9(4)V99.
KEY 0 IS p-no.
KEY "nm" IS p-name.
END
```

```
RECORD component.
FILE IS compon  RELATIVE.
02 component-key.
  04 part-no       PIC 9(4).
  04 compon-no     PIC 9(4).
KEY "pn" IS part-no.
KEY "cn" IS compon-no.
END
```

You could produce a report that lists each part and its main components by opening copies of these record descriptions and linking them appropriately. For example, you could:

1. Open the original record descriptions (*newpart* and *component*) and a copy of *newpart (npart-1)*. The copy of *newpart* will be used to supply the name of the component part.

```
OPEN newpart, component;
OPEN npart-1 AS COPY OF newpart;
```

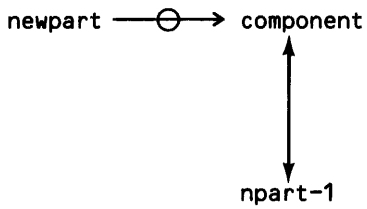
2. Establish the appropriate links between the record descriptions. The LINK OPTIONAL statement ensures that all part names (even those that do not have components) appear on the report. The LINK statement links the component number to the copy of *newpart*, thus allowing access to the name of the component.

```
LINK newpart.p-no TO OPTIONAL component.part-no;
LINK component.compon-no TO npart-1.p-no;
```

3. LIST the part name and the names of its main components.

```
LIST BY newpart.p-name HEADING "PART",
     BY npart-1.p-name HEADING "MAIN COMPONENTS";
```

A sketch of the links in the preceding query appears as:



The report produced is as follows:

```
       PART              MAIN COMPONENTS
--------------------  --------------------

16 INCH WORK CENTER   PORTABLE WORK TABLE
                      VISE
3 AMP UNIVER MOTOR
BAND SAW              3 AMP UNIVER MOTOR
                      BAND SAW BLADE
                      ROUTER AND SHAPER GD
BAND SAW BLADE
BRACES
CASTERS
MITRE GAUGE
PORTABLE WORK BENCH   BRACES
                      CASTERS
                      PORTABLE WORK TABLE
...                   ...
```

If you are familiar with the data stored in your data base, you can request a report that lists both the main components of a part and its secondary components (the components of the main components). In fact, this detailing can continue until the report breaks each part down to its most elementary component. The preceding query could be modified to produce such a report as follows:

1. Open the original record descriptions. Use the OPEN AS COPY OF statement to make two copies of *newpart* and a single copy of *component*. (The first copy of *newpart*, *npart-1*, will supply the name of the main components. The second copy of *newpart*, *npart-2*, will supply the name of the secondary components. The copy of *component*, *cp-1*, will supply the secondary components).

```
OPEN newpart,component;
OPEN cp-1 AS COPY OF component;
OPEN npart-1 AS COPY OF newpart;
OPEN npart-2 AS COPY OF newpart;
```

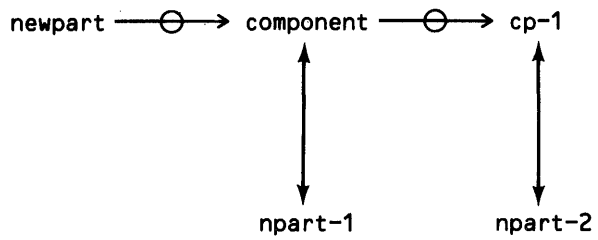
2. Link the record descriptions and their copies. The first LINK OPTIONAL statement ensures that the name of a part appears in the report even if it has no components. The second LINK OPTIONAL statement ensures that the name of a main component appears in the report even if it has no secondary components.

```
LINK newpart.p-no TO OPTIONAL component.part-no;
LINK component.compon-no TO OPTIONAL cp-1.part-no;
LINK component.compon-no TO npart-1.p-no;
LINK cp-1.compon-no TO npart-2.p-no;
```

3. List the part name, the names of its main components, and the names of its secondary components.

```
LIST BY newpart.p-name HEADING "PART",
     BY npart-1.p-name HEADING "MAIN/COMPONENTS",
        npart-2.p-name HEADING "SECONDARY/COMPONENTS";
```

A sketch of the links in this query appears as:



The report produced is:

```
                              MAIN                SECONDARY
        PART                COMPONENTS            COMPONENTS
--------------------  --------------------  --------------------

16 INCH WORK CENTER    PORTABLE WORK TABLE   CASTERS
                                             BRACES
                       VISE
3 AMP UNIVER MOTOR
BAND SAW               3 AMP UNIVER MOTOR
                       BAND SAW BLADE
                       ROUTER AND SHAPER GD
BAND SAW BLADE
...                    ...                   ...
RADIAL SAW BLADE
RADIAL SAW CENTER      RADIAL SAW            RADIAL BLADE GUARD
                                             3 AMP UNIVER MOTOR
                                             RADIAL SAW BLADE
                       SAW LEG STAND         CASTERS
                                             BRACES
...                    ...                   ...
```

SELECTING INFORMATION

The information selected for a report or a new physical file is called the target list. The target list defines the elements that appear in the target records produced by the query processor. Table 3-3 shows the statements that both identify the target list and tell ENFORM whether to produce a report or a new physical file.

Table 3-3. ENFORM Statements Used to Select Information

Statement	Function
LIST	Selects information to be printed in a report.
FIND	Selects information to be placed in a new physical file called a FIND file.

Producing a Report

Use the LIST statement when you want to select information for a report. The LIST statement selects the data base elements that contribute to the report, the data base elements that are printed in the report, and the ordering of the report lines.

Remember to establish the query environment before issuing the LIST statement and to link any necessary record descriptions either with a LINK statement or by including a WHERE clause in the LIST statement.

Permissible target list elements for a LIST statement are: field names, literals, arithmetic expressions, IF/THEN/ELSE expressions, user variables, user tables, system variables, parameters, user aggregates, or predefined aggregates. Figure 3-9 shows an ENFORM query, indicates the target list elements, and shows the report generated by the query.

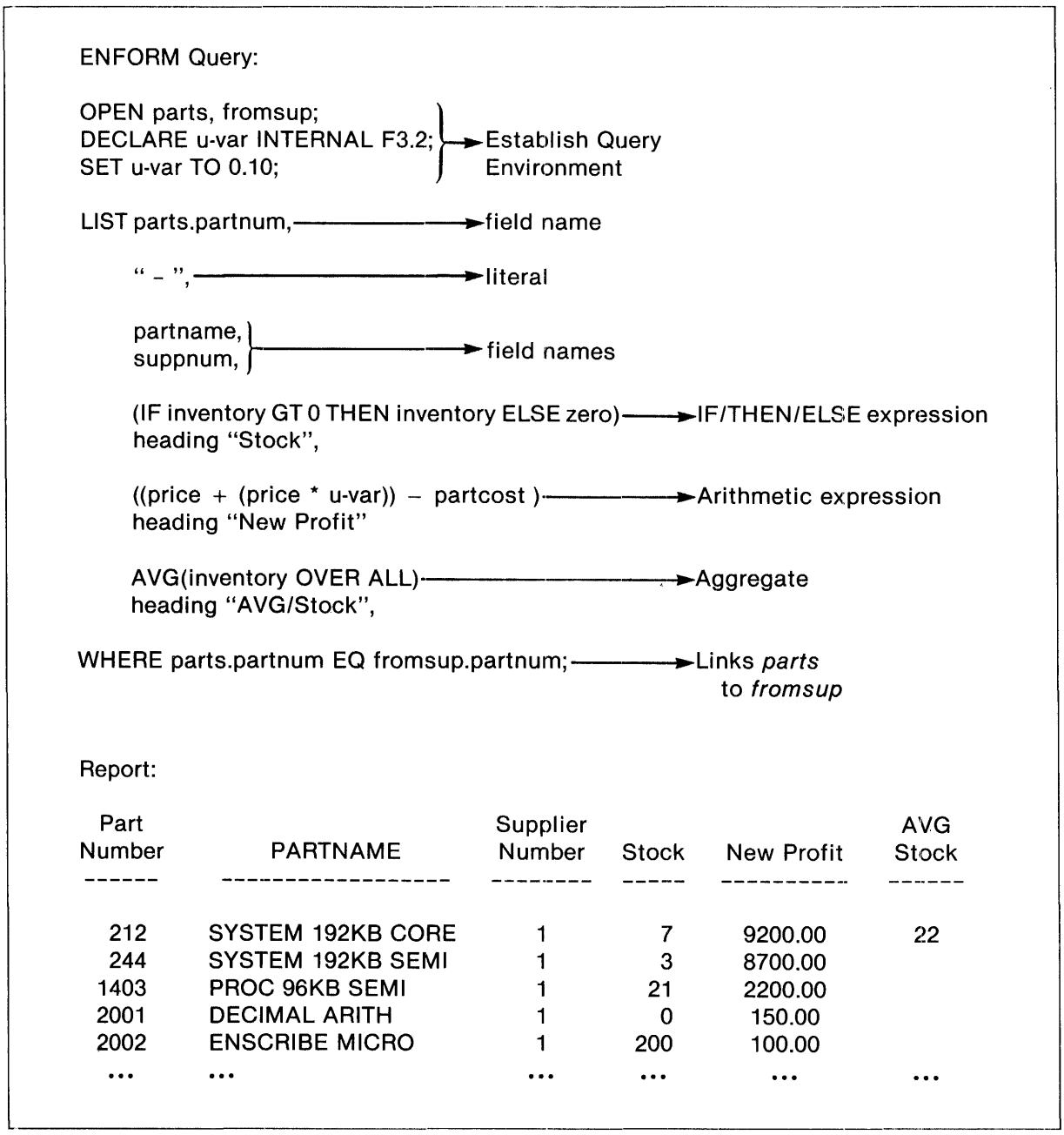


Figure 3-9. ENFORM Query and Report

Notice that the value produced by the aggregate AVG only appears on the first line of the report.

In Figure 3-9 the value of the user variable *u-var* remains constant for each target record. A user variable specified as part of a target list does not always maintain a constant value. Consider the following query where assignment syntax is used to assign a value to the user variable *user-var*:

```
OPEN order;
DECLARE user-var;
SET user-var TO 10;
LIST ordernum,
     user-var := (ordernum + user-var),
     user-var := (ordernum + user-var);
```

ENFORM determines the value of *user-var* in this query as follows:

1. ENFORM determines the value of the first arithmetic expression in the target-list. Within the arithmetic expression, *user-var* has a value of 10 (the initial value defined in the SET statement) and the value of the arithmetic expression is *ordernum* + 10.

2. ENFORM assigns the value of the arithmetic expression to the first instance of *user-var*.

3. ENFORM determines the value of the second arithmetic expression in the target-list. Within the second expression *user-var* has the value (*ordernum* + 10); therefore the value of the second arithmetic expression is *ordernum* + (*ordernum* + 10).

4. ENFORM assigns the value of the second arithmetic expression to the second instance of *user-var*.

ENFORM continues the process of re-evaluating the value of *user-var* until it encounters the end of the target-list. ENFORM repeats the process (beginning with the initial value defined in the SET statement) for every target-record.

Creating a New Physical File

Use the FIND statement to select data base elements and store them in an unstructured disc file called a FIND file. A FIND file is useful as an intermediate file for a multi-step query (one in which a LIST statement subsequently retrieves the data stored in the FIND file).

Before executing the FIND statement:

- Add a description of the record type to be created to the data dictionary.

- Establish the necessary query environment.

- Establish the necessary linking relationships either with a LINK statement or by specifying a WHERE clause in the FIND statement.

The FIND statement selects the target list elements to contribute to the FIND file. Permissible target-list elements for the input records of the FIND statement are: field names, literals, arithmetic expressions, IF/THEN/ELSE expressions, user variables, user aggregates, or predefined aggregates.

Figure 3-10 shows the DDL record description of a FIND file named *profit*, the ENFORM query containing the FIND statement, and a logical diagram of the resulting FIND file.

DDL Record Description:

```
Record profit.
  file is "profit".
  02 partnum        PIC 9(4).
  02 suppnum        PIC 9(3).
  02 partname       PIC X(18).
  02 stock          PIC 9(4).
  02 new-prof       PIC 999999V99.
  02 avg-stock      PIC 9(4).
end
```

ENFORM Query:

```
OPEN parts,fromsup,profit;
DECLARE u-var INTERNAL F3.2;
SET u-var TO 0.10;

FIND profit
        ( parts.partnum,
          parts.partname,
          fromsup.suppnum,

          stock := (IF inventory GT 0 THEN inventory ELSE zero),


          new-prof:= ((price + (price * u-var)) - partcost),


          avg-stock := AVG(inventory OVER ALL),)


WHERE parts.partnum EQ fromsup.partnum;
```

Logical Diagram of FIND File:

partnum	suppnum	partname	stock	new-prof	avg-stock
0212	001	SYSTEM 192KB CORE	0007	00920000	0022
0244	001	SYSTEM 192KB SEMI	0003	00870000	
1403	001	PROC 96KB SEMI	0021	00220000	
2001	001	DECIMAL ARITH	0000	00015000	
2002	001	ENSCRIBE MICRO	0200	00010000	
...	...	....	...	....	

Figure 3-10. DDL Record Description, ENFORM Query, and FIND File

Notice in Figure 3-10, *avg-stock* is blank except in the first record.

Using a FIND file as an intermediate file is useful when your data base contains unnormalized data. For example, suppose the following record description describes record occurrences stored in your data base:

```
RECORD alias.
FILE IS "$mkt.sample.alias" RELATIVE.
  02 name                  PIC X(20).
  02 synonym               PIC X(10) OCCURS 3 TIMES.
END
```

The OCCURS clause causes the data stored for this record description to be unnormalized. When subscripts (synonym [1], synonym [2],...) are used, ENFORM obtains the data and prints it in three columns. If you want the data to appear in one column, include a FIND file record description that associates one instance of the nonrepeating field (in this case: *name*) with each instance of the repeating group (in this case: *synonym*). For example:

```
RECORD interim.
FILE IS "$mkt.sample.normal" UNSTRUCTURED.
  02 name1           PIC X(20).
  02 synonym1        PIC X(10).
  02 name2           PIC X(20).
  02 synonym2        PIC X(10).
  02 name3           PIC X(20).
  02 synonym3        PIC X(10).
END
```

After including the record description in the dictionary, issue a FIND statement to store the data in the FIND file. For example:

```
FIND interim
  (name1:= name,
   synonym1 := synonym [1],
   name2 := name,
   synonym2 := synonym [2],
   name3 := name,
   synonym3 := synonym [3] );
```

Once the FIND statement is issued, add another record description to the dictionary that in effect breaks down each record in the FIND file so that the data appears normalized. For example:

```
RECORD normal.
FILE IS "$mkt.sample.normal" UNSTRUCTURED.
  02 name          PIC X(20).
  02 synonym       PIC X(10).
END
```

Notice that the same physical file is specified for both *interim* and *normal*. Specifying more than one record description for the same physical file allows a different view of the data. Associating more than one record description with the same physical file is only possible for unstructured file types.

RESTRICTING SELECTED INFORMATION

Restrict the information selected for the report or the FIND file by using the WHERE clause. The WHERE clause allows you to define a condition or a set of conditions that a data base record must meet before it is selected to contribute to the output record. The WHERE clause in the following query causes ENFORM to select only those record occurrences from *employee* whose salary field contains a value greater than 35000:

```
OPEN employee;
LIST empname, job, salary
     WHERE salary GREATER THAN 35000;
```

Report:

```
     EMPNAME            JOB          SALARY
-----------------  ---------------  -------

ROGER GREEN         MANAGER           39500
JERRY HOWARD        MANAGER           37000
JACK RAYMOND        MANAGER           36000
THOMAS RUDLOFF      MANAGER           38000
      .
      .
      .
```

When a user variable is used in a WHERE clause to restrict the record occurrences retrieved, the query processor always uses either the initial value (defined in a SET statement) or the default value of zero (if no SET statement exists for the user variable) for the value of the user variable. For example, consider the following query:

```
OPEN order;
DECLARE user-var;
SET user-var TO 10;
LIST ordernum,
     user-var :=(ordernum + user-var)
     WHERE user-var > 10;
```

This query always returns zero target records even though in every target record the value of user-var (ordernum + 10) is greater than 10. ENFORM does not return any target records because it uses the initial value (10) of user-var for the WHERE clause; therefore user-var is always equal to 10. If the WHERE clause contains:

```
WHERE user-var = 10
```

all the records in *order* are returned because the value of user-var in the WHERE clause is always equal to 10.

SORTING AND GROUPING SELECTED INFORMATION

Sorting the information selected for a report makes the report easier to read. Grouping the sorted information improves the appearance of the report and more clearly defines important data.

Sorting the information selected for a FIND file eliminates the need for sorting the records when the FIND file is used in a subsequent query. (Remember, the SEQUENCE IS clause must be used in the DDL record description of the FIND file.)

Table 3-4 shows the ENFORM clauses used to group and sort information.

Table 3-4. Clauses Used To Group and Sort Information

Clause	Function
BY	Groups and sorts target-records in ascending order according to the value of a field.
BY DESC	Groups and sorts target-records in descending order according to the value of a field.
ASCD	Sorts target-records in ascending order according to the value of a field.
DESC	Sorts target-records in descending order according to the value of a field.

Use the BY and BY DESC clauses to group and sort field values. Field names grouped and sorted by BY and BY DESC clauses are called by-items. If a BY or BY DESC clause is specified in a LIST statement, the printing of all duplicate values is suppressed. For example, consider the following query and report:

```
OPEN odetail;
LIST BY ordernum,
        partnum;
```

Report:

```
            Part
ORDERNUM  Number
--------  ------

      21     244
            2001
            2403
            4103
      25     244
            5103
            6301
            6402
      30     244
            2001
     ...    ...
```

Notice that only the first instance of a value for *ordernum* appears in the report.

If a BY or BY DESC clause is specified in a FIND statement, duplicate values are included in the FIND file. For instance, consider the query and the diagram of a FIND file shown in Figure 3-11.

ENFORM Query:

```
OPEN odetail,orderup;
FIND orderup
     (BY odetail.ordernum,
         odetail.partnum );
```

FIND File Diagram:

Ordernum	Partnum
0021	0244
0021	2001
0021	2403
0021	4103
0025	0244
...	...

Figure 3-11. Sample ENFORM Query and FIND File Diagram

Notice that all the values of *ordernum* appear in the FIND file.

When multiple BY, BY DESC, ASCD, and DESC clauses appear in the same query, ENFORM establishes a major to minor sort sequence. The first field name modified by a BY, BY DESC, ASCD, or DESC clause has first priority, the second next priority, and so on, down to the last. In the following example, *regnum* is sorted and grouped first, *branchnum* is sorted next, and *job* is sorted and grouped last:

```
OPEN employee;
LIST BY DESC regnum,
BY DESC branchnum,
BY job;
```

Report:

```
REGNUM   BRANCHNUM    JOB
------  ---------  -------

    99          1  MANAGER
     5          3  MANAGER
     2          1  MANAGER
                   SALESMAN
     1          2  SALESMAN
                1  MANAGER
                      ...
```

SPECIFYING COMPUTATIONS FOR A REPORT

To prepare a complete report, arithmetic operations are often necessary. Besides arithmetic expressions and aggregates (refer to the *ENFORM Reference Manual* for information about these language elements), ENFORM also provides clauses that allow you to:

- Calculate a total or subtotal value for numeric elements.
- Calculate a percentage value for numeric elements.
- Calculate a running total for the values of a numeric element based either on all values or the grouped values of the element.

Table 3-5 shows the ENFORM clauses that can be used only with the LIST statement to specify arithmetic operations for a report.

Table 3-5. Clauses Used to Specify Computations

Clause	Function
SUBTOTAL	Generates a subtotal for an element.
TOTAL	Generates a total for an element.
PCT	Generates a percentage of the total for an element.
CUM	Generates a running total for an element.

When specifying any of the clauses shown in Table 3-5, consider the number of digits you expect to be returned. If that number exceeds the number of digits defined in the dictionary for the field over which the calculation is performed, an overflow condition results. When an overflow condition occurs, overflow characters (asterisks by default) appear on the report instead of the figure you expect. To avoid this situation, use an AS clause (described later in this section) to modify the element. The AS clause can increase the number of digits displayed.

When a reference to a user variable is modified by any of these clauses, an overflow condition does not result in the printing of overflow characters.

Calculating a Subtotal

Use the SUBTOTAL clause to generate a subtotal for the values of a numeric element within one or more by-items. When the value of a by-item changes, ENFORM prints the subtotal on the report under the column of the element being subtotaled. ENFORM marks the subtotal with a subtotal string (the default is an asterisk *) in the by-item column to which the subtotal relates.

Write the SUBTOTAL clause after the element whose values are to be subtotaled. Unless you specifically indicate otherwise, ENFORM computes the subtotal over all the by-items in the report. The following example illustrates subtotals for the values of *salary* within the by-items *regnum* and *branchnum*:

```
OPEN employee;
LIST BY regnum,
     BY branchnum,
        job,
        salary AS I9 SUBTOTAL;
```

Report:

```
REGNUM  BRANCHNUM      JOB        SALARY
------  ---------  -----------  ---------

     1          1  MANAGER          36000
                   MANAGER          32000
                   SALESMAN         19000
                   SYS.-ANAL.       25000
                   SECRETARY        12000
                   SALESMAN         26000
                                 ---------
            *                      150000

                2  MANAGER          37000
                   SALESMAN         30000
            *                    ---------
                                      67000

*                                ---------
                                    217000
...        ...          ...           ...
```

When you include the OVER syntax with the SUBTOTAL clause, ENFORM calculates a subtotal over a named by-item. In the following example, the SUBTOTAL clause generates a subtotal for the values of *quantity* over the by-item *ordernum*:

```
OPEN odetail;
LIST BY ordernum,
        partnum,
        quantity SUBTOTAL OVER ordernum;
```

Report:

```
            Part
ORDERNUM    Number   QUANTITY
--------    ------   --------

      21      244           1
              2001          2
              2403          2
              4103          2
                       --------
   *                          7
   ...        ...         ...
```

Calculating a Total

Use the TOTAL clause to generate a grand total for the values of a numeric element. The following example generates a grand total for the values of *quantity*:

```
OPEN odetail;
LIST BY ordernum,
        partnum
        quantity TOTAL,
     WHERE ordernum LE 21;
```

Report:

```
            Part
ORDERNUM    Number   QUANTITY
--------    ------   --------

      21      244           1
              2001          2
              2403          2
              4103          2
                       --------
                       --------
                              7
```

Calculating Percentages

Use the PCT clause to calculate percentages for numeric elements. The element for which the percentage is being calculated need not appear in the report. The following example shows the PCT clause modifying *quantity*. ENFORM assumes that the percentage is to be calculated OVER ALL even though the OVER ALL syntax is omitted.

```
OPEN odetail;
LIST BY ordernum,
        partnum,
        quantity,
        quantity PCT AS F5.2;
```

Report:

```
            Part                  PCT
ORDERNUM  Number  QUANTITY  QUANTITY
--------  ------  --------  --------

      21     244         1      4.76
            2001         2      9.52
            2403         2      9.52
            4103         2      9.52
      25     244         1      4.76
            5103         1      4.76
            6301         2      9.52
            6402        10     47.62
```

When the OVER ALL syntax is specified or assumed, ENFORM calculates percentage values by performing the following:

1. ENFORM adds all of the values of the element modified by the PCT clause together to obtain a total. In the preceding example these values are: 1, 2, 2, 2, 1, 1, 2, and 10. The total of these values is 21.

2. ENFORM then divides the value for each line of the report by the total to obtain the percentage value. In the preceding example, the *quantity* value appearing on the first line of the report is 1. Dividing 1 by 21 (the total obtained in step 1) results in a percentage value of 4.76.

The following example shows the PCT clause with the OVER syntax.

```
OPEN odetail;
LIST BY ordernum,
        partnum,
        quantity,
        quantity PCT OVER ordernum,
      WHERE ordernum LE 25;
```

Report:

ORDERNUM	Part Number	QUANTITY	PCT QUANTITY	
21	244	1	14.29	
	2001	2	28.57	
	2403	2	28.57	Percent of order 21
	4103	2	28.57	
25	244	1	7.14	
	5103	1	7.14	Percent of order 25
	6301	2	14.29	
	6402	10	71.43	

When the OVER syntax is specified, ENFORM calculates grouped percentage values by performing the following:

1. ENFORM adds together the grouped values of the element modified by the PCT clause to obtain a total grouped value. (Remember an element is grouped when a preceding element in the query is modified by a BY or BY DESC clause.) In the preceding example, the first grouped *quantity* values are: 1, 2, 2, and 2 resulting in a total grouped value of 7. ENFORM calculates the total grouped value of each group in the report.

2. ENFORM then divides the individual values for each group by the total grouped value to obtain the group percentage values. In the preceding example, the first *quantity* value of the first grouped value is 1. ENFORM divides 1 by the total grouped value 7 (obtained in the first step) resulting in a grouped percentage value of 14.29.

Combine the SUBTOTAL and TOTAL clauses with the PCT clause to obtain the subtotal and total of the percentage values. The PCT values do not total exactly one hundred per cent since precision is lost due to truncation during division. In the following example, the SUBTOTAL and TOTAL clauses are used with the PCT clause to modify *quantity*:

```
OPEN odetail;
LIST BY ordernum,
        partnum,
        quantity,
        quantity PCT AS F5.2,
        SUBTOTAL OVER ordernum,
        TOTAL,
    WHERE ordernum LT 30;
```

Report:

ORDERNUM	Part Number	QUANTITY	PCT QUANTITY
21	244	1	4.76
	2001	2	9.52
	2403	2	9.52
	4103	2	9.52
*			33.32
25	244	1	4.76
	5103	1	4.76
	6301	2	9.52
	6402	10	47.62
*			66.66
			99.98

Generating a Running Total

Use the CUM clause to generate a running total for the values of the numeric element it modifies based either on all instances of the element or on the instances of the element grouped within a by-item. The following query generates a running total based on all the instances of *quantity*:

```
OPEN odetail;
LIST BY ordernum,
        partnum,
        quantity,
        quantity CUM AS I7,
    WHERE ordernum LE 25;
```

Report:

```
            Part                CUM
ORDERNUM  Number  QUANTITY  QUANTITY
--------  ------  --------  --------

      21     244         1         1
            2001         2         3
            2403         2         5
            4103         2         7
      25     244         1         8
            5103         1         9
            6301         2        11
            6402        10        21
```

The following query generates a running total for *quantity* within the by-item *ordernum*:

```
OPEN odetail;
LIST BY ordernum,
        partnum,
        quantity,
        quantity CUM OVER ordernum AS I7,
        WHERE ordernum LE 25;
```

Report:

```
            Part                CUM
ORDERNUM  Number  QUANTITY  QUANTITY
--------  ------  --------  --------

      21     244         1         1
            2001         2         3
            2403         2         5
            4103         2         7
      25     244         1         1
            5103         1         2
            6301         2         4
            6402        10        14
```

Running total begins anew at each new order.

FORMATTING A REPORT

If you specify a LIST statement in your query, ENFORM selects information from your data base, formats the information, and directs the information to the spooler, a physical file, or your terminal screen. If you write a query that consists of only an OPEN statement, a DECLARE statement, and a LIST statement with nothing but field names, arithmetic expressions, IF/THEN/ELSE expressions, user variables, user tables, user aggregates, or predefined aggregates (no by-items, no clauses, no options set), ENFORM writes the selected information according to the following defaults:

- Target List Elements

  ENFORM prints the first target list element specified in the LIST statement in the leftmost report column, the second in the next column, and so on. If all the needed columns do not fit on a line, ENFORM folds the line including the heading.

- Data order

  ENFORM determines the order in which data values are printed within a column by the order in which the data values are read from the data base. ENFORM does not sort the data values.

- Column width

  ENFORM determines the width of a column by whichever is larger: the length of an element in characters (as described in the data dictionary or the DECLARE statement) or the length of the element's heading in characters.

- Horizontal spacing

  ENFORM skips two spaces between columns.

- Vertical spacing

  ENFORM uses single spacing between report lines.

- Margins

  ENFORM prints the first column in line column 1. ENFORM determines the location of the right margin by calculating the total number of characters in each column plus the two spaces between each column.

- Line length

  ENFORM uses a maximum line length of 132 characters for a printer; it uses a maximum line length of 80 characters for a terminal.

- Page length

  ENFORM prints sixty lines on a page beginning at top-of-form.

- Page numbers

  ENFORM does not print page numbers.

- Titles

  ENFORM does not print any titles.

- Headings

  If a heading is defined for the element in either the data dictionary or the DECLARE statement, ENFORM prints that heading. If not, ENFORM prints the name of the element as the heading. If neither applies (the element is a predefined or user aggregate), ENFORM creates a heading. ENFORM prints the values of arithmetic expressions without headings.

- Data justification

  ENFORM prints numeric data as right-justified within the column; it prints alphanumeric data as left-justified within the column.

- Display format

  ENFORM determines the display format for fields by the record description entry in the dictionary. ENFORM displays user variables, user table elements, user aggregates, and predefined aggregates as 14 character integers. ENFORM displays the result of arithmetic expressions in 14 character fields, as either integer or fixed. ENFORM determines the default display format for IF/THEN/ELSE expressions by the elements in the expressions.

The ENFORM defaults might be satisfactory for some of your reports, but other reports might require particular arrangements of the report data. The way your report looks can be affected by the addition of ENFORM statements or ENFORM clauses that:

- Add information to a report.
- Define the actual layout of the report.
- Define the display format of the elements in the report.

Printing Information Within a Report

Adding information such as titles, subtitles, and footings to a report improves the appearance of the report and provides documentation as to the time, date, and reason for the report. ENFORM provides statements and clauses that allow you to add information:

- At the beginning or end of a report.
- Within the body of a report.
- At the end of every report page.
- At the beginning of every report page.

Table 3-6 shows the ENFORM statements and clauses that can be used to add information to a report.

Table 3-6. Statements and Clauses Used To Add Information to Reports

Statement	Clause	Function
	AFTER CHANGE	Prints information preceding the records for each by-item.
AT END	AT END PRINT	Prints information at the end of a report.
AT START	AT START PRINT	Prints information at the beginning of a report.
	BEFORE CHANGE	Prints information following the records for each by-item.
FOOTING	FOOTING	Prints a footing at the bottom of each page.
SUBFOOTING	SUBFOOTING	Prints a subfooting at the bottom of the page preceding the footing.
SUBTITLE	SUBTITLE	Prints a subtitle at the top of the page following the title.
TITLE	TITLE	Prints a title at the top of a report page.

The statements and clauses used to add information to reports obtain this information from a print list you supply. The query compiler stores the print list information in its internal table. The internal table contains an entry for each element that exists in the print list. If the internal table overflows, remove all session-wide declarations overridden in the current report by issuing a session-wide statement (for example AT START or TITLE) without the print list parameter.

To conserve space in the internal table, use TAB instead of SPACE in the print list to obtain the necessary report format. The query compiler does not allocate table space for a TAB clause whereas it does allocate table space for a SPACE clause. Judicious use of string literals also saves table space. For example: "PHONE: " is equivalent to "PHONE:", SPACE 3, but the former causes only one table entry while the latter causes two table entries.

Figure 3-12 shows where the information supplied by these statements and clauses prints in a report.

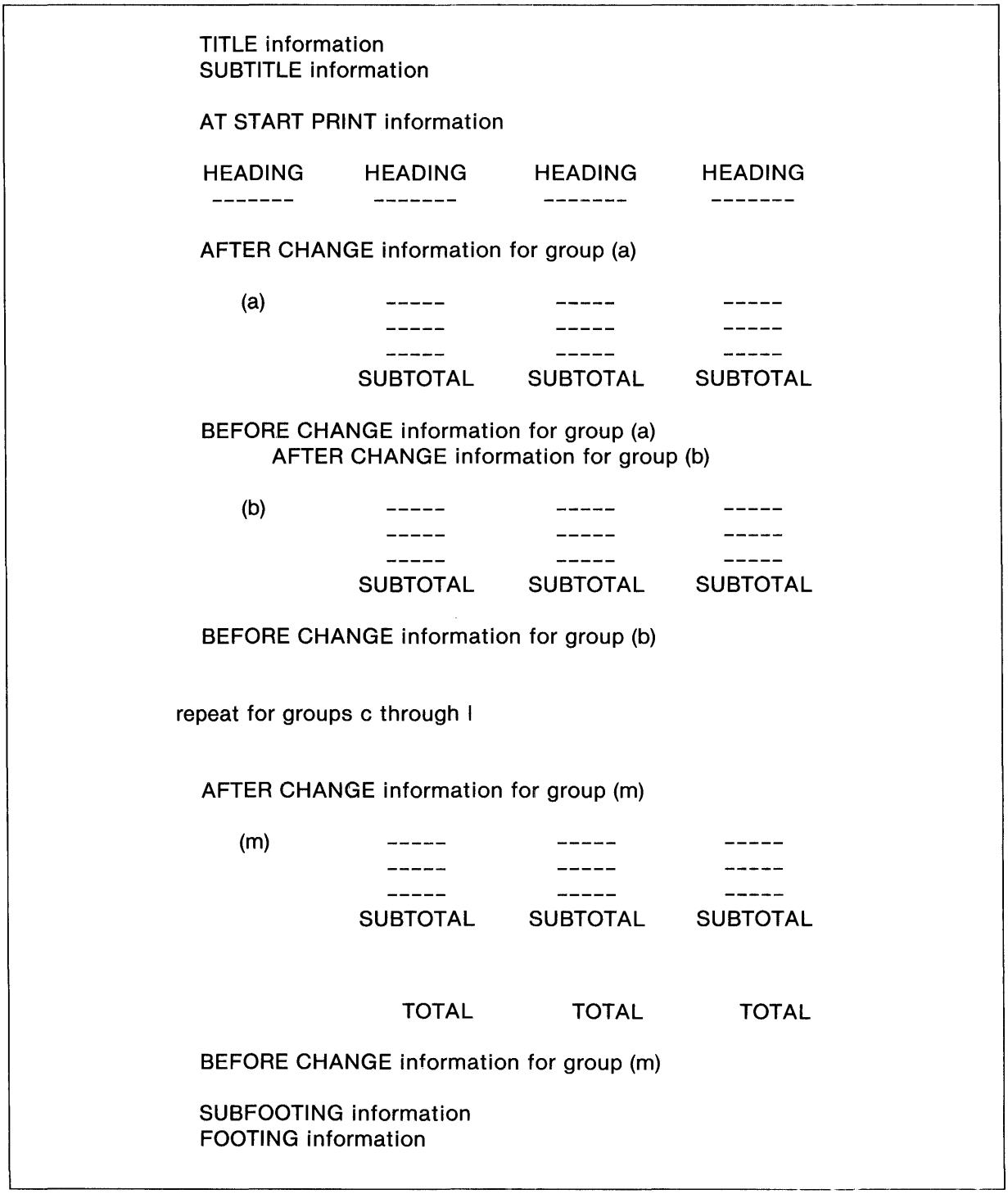


Figure 3-12. ENFORM Report Format

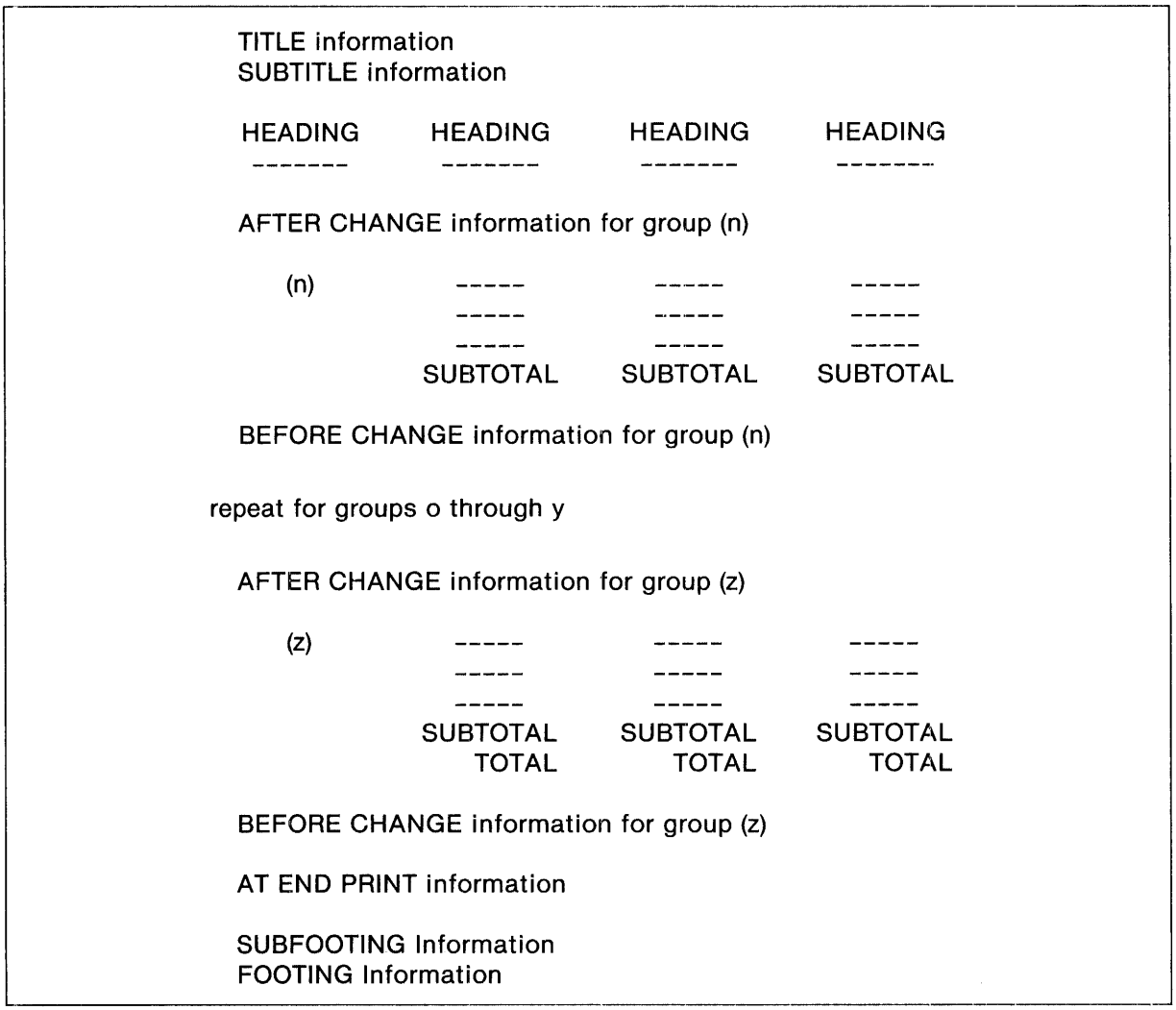


Figure 3-12. ENFORM Report Format (Continued)

ADDING INFORMATION WITHIN THE BODY OF THE REPORT. Use either the BEFORE CHANGE or AFTER CHANGE clause to specify printing of information between the end of printing of one set of grouped values and the start of the next set of grouped values. (Remember values are grouped when a field name is modified by a BY or BY DESC clause.)

The BEFORE or AFTER keywords of these clauses refer to the values printed, not to the location of the printed information:

- The BEFORE CHANGE clause obtains values to print from the last set of grouped values printed; that is, the last value of the element before the by-item value changes.
- The AFTER CHANGE clause obtains values to print from the next set of grouped values to be printed; that is, the first value of the element after the by-item value changes.

The following example shows the use of both the AFTER CHANGE and BEFORE CHANGE clauses:

```
DECLARE order-sum AS M<ZZZ,ZZZ,ZZ9.99>;
OPEN odetail,parts;
LINK odetail TO parts VIA partnum;
LIST BY ordernum, odetail.partnum,
        (price * quantity) AS M<ZZZ,ZZZ,ZZ9.99>,
        order-sum := SUM(price * quantity) OVER ordernum),
                     NOPRINT,
  WHERE ordernum LE 25,
  BEFORE CHANGE ON ordernum PRINT
         "Before change on order " ordernum
         " total order revenue is " order-sum,
  AFTER CHANGE ON ordernum PRINT
         "After Change on order " ordernum
         " total order revenue is " order-sum;
```

Report:

```
            Part
ORDERNUM  Number
--------  ------

After Change on order 21 total order revenue is      158,200.00
      21     244       87,000.00
            2001        3,000.00
            2403       19,200.00
            4103       49,000.00
Before change on order 21 total order revenue is     158,200.00
After change on order 25 total order revenue is      115,800.00
      25     244       87,000.00
            5103        8,000.00
            6301        5,800.00
            6402       15,000.00
Before change on order 25 total order revenue is     115,800.00
```

PRINTING INFORMATION AT THE BEGINNING OR END OF A REPORT. Specify AT START or AT END to print one or more lines of user-defined information at the beginning (AT START) or at the end (AT END) of a report. The session-wide AT START statement prints information directly after the title (if any) on the first page of every report in the current ENFORM session unless cancelled, reset, or overridden. The AT END statement prints information on the last page of every report in the current ENFORM session unless cancelled, reset, or overridden. Refer to the *ENFORM Reference Manual* for information about cancelling or resetting these statements. The AT START and AT END clauses temporarily override the AT START and AT END statements respectively. Consider the following queries:

```
AT START PRINT "Confidential Report";
AT END PRINT "End of Confidential Report";
OPEN employee;
LIST empname, salary;
CLOSE employee;
OPEN parts;
LIST partnum,partname,price,
  AT START PRINT "Parts Report",
AT END PRINT "End of Parts Report";
```

Report generated by first query:

```
Confidential Report

     EMPNAME         SALARY
------------------  ------

ROGER GREEN          39500
JERRY HOWARD         37000
JACK RAYMOND         36000
...                  ...

End of Confidential Report
```

Report generated by second query:

```
Parts Report

Part
Number        PARTNAME          PRICE
------  ------------------  ---------

   212  SYSTEM 192KB CORE    92000.00
   244  SYSTEM 192KB SEMI    87000.00
  1403  PROC 96KB SEMI       22000.00
  ...    ...                  ...
  7301  POWER MODULE          2400.06

End of Parts Report
```

PRINTING INFORMATION AT THE BOTTOM OF EVERY REPORT PAGE. Use either SUBFOOTING or FOOTING to specify printing of information at the bottom of each page of a report. If both are specified, the information specified for FOOTING is printed below the information specified for SUBFOOTING.

The FOOTING and SUBFOOTING statements are session-wide; they apply to all reports generated during the current ENFORM session unless they are cancelled, reset, or overridden. The SUBFOOTING and FOOTING clauses temporarily override the SUBFOOTING and FOOTING statements for the report generated by the associated LIST statement.

The following query contains both a FOOTING and SUBFOOTING statement.

```
TITLE " Order Summary";
SUBFOOTING "Summary of";
FOOTING "Orders For " ordernum;
OPEN order,odetail,parts;
LINK order TO odetail VIA ordernum;
LINK parts TO odetail VIA partnum;
LIST BY order.ordernum, FORM,
        custnum, parts.partnum,
        quantity,price;
```

Report:

```
Order Summary

                    Part
ORDERNUM  CUSTNUM  Number  QUANTITY   PRICE
--------  -------  ------  --------  ---------

      21     1234     244         1   87000.00
                     2001         2    1500.00

Summary of
Orders For  21

 Order Summary

                    Part
ORDERNUM  CUSTNUM  Number  QUANTITY   PRICE
--------  -------  ------  --------  ---------

      25     7777     244         1   87000.00
                     6301         2    2900.00
                     6402        10    1500.00

Summary of
Orders For  25
...
```

PRINTING INFORMATION AT THE TOP OF EACH REPORT PAGE. Use the SUBTITLE and TITLE statements or clauses to specify printing of one or more text lines at the top of each report page. The difference between SUBTITLE and TITLE is that the user-supplied text specified in a TITLE statement or clause prints before the user-supplied text specified in a SUBTITLE statement or clause. The text supplied in a SUBTITLE statement or clause prints whether or not a TITLE statement or clause is specified.

A SUBTITLE or TITLE statement is in effect session-wide; that is, it applies to all the reports generated during the current ENFORM session unless cancelled, reset, or overridden. A SUBTITLE or TITLE clause overrides the SUBTITLE or TITLE statement respectively.

In the following queries, titles are printed for two reports:

```
OPEN parts;
TITLE "Daily Inventory Report for " @DATE AS DATE *;
SUBTITLE "Location " location " only";
LIST BY partnum, partname,inventory,
     WHERE location is "L98";
CLOSE parts;
OPEN employee;
LIST BY regnum, By branchnum, salary,
        AS M<ZZZ,ZZZ,ZZ9.99>,
     WHERE regnum IS 1 AND branchnum IS 2,
     TITLE TAB 4 "*** CONFIDENTIAL REPORT ***", CENTER,
           SKIP 2,
     SUBTITLE "Region Salary Report as of "
        @DATE AS DATE *, SKIP 1,
        "For Region " regnum CENTER;
```

Report generated by the first LIST statement:

```
Daily Inventory Report for 05/24/82

        Location L98 only

 Part
Number       PARTNAME        INVENTORY
------  ------------------   ---------

  5103  MAG TAPE DR 8/16             8
```

Report generated by the second LIST statement:

```
      *** CONFIDENTIAL REPORT ***

  Regional Salary Report as of 05/24/82
             For Region 1

REGNUM  BRANCHNUM       SALARY
------  ---------  --------------

    1           2       37,000.00
                        30,000.00
```

Defining the Layout of the Report

Defining the layout of the report improves its appearance and readability. ENFORM provides clauses that allow you to:

- Center one or all of the elements within the report.
- Paginate the report.
- Change the default column headings of the report.
- Suppress the printing of a column heading within the report.
- Suppress the printing of both the column heading and an element within the report.
- Tell ENFORM to begin a new report line.
- Set a tab for the report.

Table 3-7 shows the clauses that allow you to define the report layout.

Table 3-7. Clauses that Define Report Layout

Clause	Function
CENTER	Centers an object within its context.
FORM	Controls when to skip to a new page.
HEADING	Overrides a default column heading.
NOHEAD	Suppresses printing of a column heading.
NOPRINT	Suppresses printing of an element and its heading.
SKIP	Specifies vertical spacing.
SPACE	Specifies horizontal spacing.
TAB	Specifies the column in report to begin printing.

CENTERING ONE ELEMENT OR ALL ELEMENTS OF A REPORT. Use the CENTER clause to center either an element under its column heading or a column heading over the associated element. The object centered depends on which is longer: the element or the heading. Specify the CENTER clause after the element you want centered. In the following example, the CENTER clause centers *inventory*:

```
OPEN parts;
LIST partnum, partname, location CENTER;
```

Report:

```
 Part
Number       PARTNAME        LOCATION
------  ------------------   --------

  212   SYSTEM 192KB CORE       J87
  244   SYSTEM 192KB SEMI       B78
 1403   PROC 96KB SEMI          A21
 2001   DECIMAL ARITH           X10
 2002   ENSCRIBE MICRO          X11
  ...   ...                     ...
```

If all the elements in a LIST statement are to be centered, use the CENTER clause after the last target list element and specify CENTER ALL:

```
OPEN parts;
LIST partnum,partname,inventory,
     location, CENTER ALL;
```

Report:

```
 Part
Number       PARTNAME          INVENTORY  LOCATION
------  -------------------    ---------  --------

  212   SYSTEM 192KB CORE           7         J87
  244   SYSTEM 192KB SEMI           3         B78
 1403   PROC  96KB SEMI            21         A21
 2001   DECIMAL ARITH             100         X10
 2002   ENSCRIBE MICRO            200         X11
```

The elements in the preceding example are centered although they might not appear so to the casual viewer. ENFORM supports only fixed length strings. Consider *partname*. The dictionary record description of *partname* is PIC X(18). When ENFORM centers the values of *partname*, it uses the fixed length of 18 characters. ENFORM left-justifies the values within this 18-character field. ENFORM uses the default heading PARTNAME since no heading is specified in the query. Because the heading is only 8 characters long, ENFORM centers the heading.

PAGINATING A REPORT. Use the FORM clause to determine report pagination. The FORM clause can either modify a by-item or stand alone as a target list element.

When a digit is included with a FORM clause modifying a by-item, ENFORM generates a page break if the number of lines indicated by the digit is not available when the new by-item begins. A page break is not generated from each by-item change. In the following example, a page break is generated only if fewer than 4 lines remain on the page when a new *custnum* is to be printed:

```
OPEN custnum;
LIST BY custnum FORM 4,...;
```

If the FORM clause modifies a by-item and the digit is omitted from the clause, a page break occurs every time the by-item changes value. The following example causes ENFORM to print each distinct *custnum* on a separate page.

```
OPEN customer;
LIST BY custnum FORM,...;
```

When a FORM clause appears as part of the target list, a page break occurs when FORM is encountered. A number following FORM has no meaning in this case:

```
OPEN customer;
LIST FORM, custnum,...;
```

CHANGING THE DEFAULT COLUMN HEADING. Use the HEADING clause to temporarily change the default column heading. The following example changes the default column heading for *ordernum:*

```
OPEN order;
LIST ordernum,HEADING "ORDER NUMBER",
     salesman
     WHERE ordernum LE 30;
```

Report:

```
ORDER  NUMBER  SALESMAN
------------  --------

          21       205
          25       212
          30       222
```

Create multiple line headings by using a slash (/) within the heading string. The following example changes the default column heading for *ordernum* and splits the heading into two lines:

```
OPEN order;
LIST ordernum HEADING "ORDER/NUMBER"
     salesman,
     WHERE ordernum LE 30;
```

Report:

```
ORDER
NUMBER  SALESMAN
------  --------

    21       205
    25       212
    30       222
```

SUPPRESSING THE PRINTING OF A COLUMN HEADING. Use the NOHEAD clause to suppress the printing of a column heading. Suppressing a column heading is useful when you want to fold a line into two lines. The following example prints *state* under *city* and suppresses the printing of the column heading for *state*:

```
OPEN customer;
LIST custnum, custname,
     city HEADING "CITY/STATE"/
     TAB 31 state, NOHEAD;
```

Report:

```
                                  CITY
CUSTNUM       CUSTNAME            STATE
-------  ------------------  --------------

     21  CENTRAL UNIVERSITY  PHILADELPHIA
                             PENN
    123  BROWN MEDICAL CO    SAN FRANCISCO
                             CALIFORNIA
    143  STEVENS SUPPLY      DENVER
                             COLORADO
    324  PREMIER INSURANCE   LUBBOCK
                             TEXAS
    543  FRESNO STATE BANK   FRESNO
                             CALIFORNIA
```

SUPPRESSING THE PRINTING OF BOTH THE COLUMN HEADING AND THE ELEMENT. Use the NOPRINT clause to suppress the printing of both the column heading and the element. The element modified by a NOPRINT clause still contributes to any calculations of which it is a part.

The following example generates a report containing a list of employees with the highest salaries. Since *salary* is not for public consumption, the NOPRINT clause is used to suppress the printing of this element:

```
OPEN employee;
LIST empnum, empname,
     DESC salary NOPRINT,
     job,
     WHERE salary GT 22000;
```

Report:

```
EMPNUM        EMPNAME             JOB
------  ------------------  ----------------

     1  ROGER GREEN         MANAGER
    32  THOMAS RUDLOFF      MANAGER
    23  JERRY HOWARD        MANAGER
   ...  ...                 ...
```

INDICATING A NEW LINE. Use the SKIP clause to specify that the elements following the SKIP clause are to be printed on a new line. If a digit is included as part of the SKIP clause, ENFORM skips the number of lines indicated by the digit before printing the elements. When the digit is omitted, only one line is skipped unless the option variable @VSPACE is set.

In the following example, the SKIP clause causes ENFORM to print *state* on the next line:

```
OPEN customer;
LIST custnum,custname,
     city HEADING "CITY/STATE",
     SKIP 1,TAB 31,state, NOHEAD;
```

Report:

```
                                  CITY
CUSTNUM       CUSTNAME            STATE
-------  ------------------  --------------

     21  CENTRAL UNIVERSITY  PHILADELPHIA
                             PENN
     ...  ...                 ...
```

The SKIP clause can be affected by the option variable @VSPACE described in the *ENFORM Reference Manual.* Refer to this manual for more information.

CHANGING THE DEFAULT SPACING. Use the SPACE clause to temporarily change the default spacing between two report columns. Specification of a digit after the SPACE keyword indicates the number of spaces you want between columns. If no digit is present, ENFORM skips one space. Default spacing resumes after ENFORM prints the element associated with the SPACE clause. The following example changes the default spacing between *custname* and *city*:

```
OPEN customer;
LIST custnum, custname,
     SPACE 7, city;
```

Report:

```
CUSTNUM        CUSTNAME              CITY
-------  ------------------      --------------

     21  CENTRAL UNIVERSITY      PHILADELPHIA
    123  BROWN MEDICAL CO        SAN FRANCISCO
    143  STEVENS SUPPLY          DENVER
    324  PREMIER INSURANCE       LUBBOCK
    543  FRESNO STATE BANK       FRESNO
```

SETTING A TAB FOR A REPORT. Use the TAB clause to define where the next element is to be printed in a report line. The default column spacing resumes after the element is printed. A digit following the keyword TAB indicates the column number in which the element is printed. Remember that the digit specified in a TAB clause is always relative to the left margin setting and always on the current line even if ENFORM must backspace to reach the specified column. If no digit is specified in a TAB clause, column 1 is assumed.

In the following example, the TAB clause causes ENFORM to print *city* in column 50:

```
OPEN customer;
LIST custnum, custname,TAB 50, city;
```

Report:

```
CUSTNUM        CUSTNAME                        CITY
-------  ------------------              --------------

     21  CENTRAL UNIVERSITY              PHILADELPHIA
    123  BROWN MEDICAL CO                SAN FRANCISCO
    143  STEVENS SUPPLY                  DENVER
    ...  ...                             ...
```

Formatting The Appearance Of Selected Information

Formatting the information you select for your report makes the report both useful and visually pleasing. ENFORM provides clauses that allow you to:

- Temporarily change the display format of an element.
- Print a time value on a report.
- Print a data value on a report.

Table 3-8 shows these clauses.

Table 3-8. Clauses Used To Format Selected Information

Clause	Function
AS	Temporarily overrides the display format for printing the value of a element.
AS DATE	Temporarily converts a date from internal format to a display format for printing.
AS TIME	Temporarily converts a time from internal format to a display format for printing.

TEMPORARILY CHANGING THE DEFAULT DISPLAY FORMAT OF AN ELEMENT. Use the AS clause to temporarily override the default display format (described earlier in this section) used for printing the value of an element. Within the AS clause, indicate the new display format by specifying edit descriptors, modifiers, and decorations.

Edit descriptors specify the display format as alphanumeric, fixed, or integer. A special edit descriptor describes the display format according to a template. Edit descriptors also specify the display format width. The following query indicates the display format for *price* is fixed with 5 digits to the left of the decimal place and 2 digits to the right of the decimal place:

```
OPEN parts;
LIST partnum,
     partname,
     price AS F7.2,
   WHERE partnum LT 1000;
```

Report:

```
 Part
Number        PARTNAME        PRICE
------  ------------------  --------

   212  SYSTEM 192KB CORE   97000.00
   244  SYSTEM 192KB SEMI   87000.00
```

Edit modifiers indicate field blanking, character insertion, fill character specification, scale factor specification, left and right justification, overflow character specification, and symbol substitution. The following query indicates *partname* is to be printed on the report right-justified (left-justified is the default for alphanumeric data) in a column 20 characters wide:

```
OPEN parts;
LIST partnum,
     partname AS "[RJ]A20",
     price,
  WHERE partnum LT 1000;
```

Report:

```
 Part
Number        PARTNAME           PRICE
------  --------------------  ---------

   212     SYSTEM 192KB CORE   97000.00
   244     SYSTEM 192KB SEMI   87000.00
```

Decorations specify a character string that can be added to a column on the report, the conditions under which the string is to be added, the location of the string, and whether it is to be added either before or after the other formatting is done. The following query specifies that when the value of *price* is negative, zero, or positive, a dollar sign ($) is to be inserted in the first position immediately to the left of the value:

```
OPEN parts;
LIST partnum,
     partname,
     price AS "[MZPF '$']F8.1",
 WHERE partnum LT 1000;
```

Report:

```
 Part
Number     PARTNAME          PRICE
------  ------------------  --------

   212  SYSTEM 192KB CORE   $97000.0
   244  SYSTEM 192KB SEMI   $87000.0
```

Refer to the description of the AS clause in the *ENFORM Reference Manual* for more information about display formatting.

PRINTING A DATE VALUE ON A REPORT. Use the AS DATE clause to specify printing of a date that is stored in internal format. If dates are not stored in internal format, use the JULIAN-DATE clause to convert them to internal format.

In the following example, the AS DATE clause prints the current date using by the default date format to modify the system variable @DATE:

```
OPEN orders;
LIST ordernum,
     TITLE "ORDER NUMBERS" SPACE 10,
     @DATE AS DATE *, SKIP 2;
```

Report:

```
ORDER NUMBERS          05/07/82

ORDERNUM
--------

      21
      25
     ...
```

To print a date that is not stored in internal format, use the AS DATE clause to modify the JULIAN-DATE clause:

```
OPEN order;
LIST ordernum,
     JULIAN-DATE(oyear,omonth,oday) AS DATE *;
```

Report:

```
ORDERNUM
--------

      21  01/10/78
      25  01/23/78
    ...     ....
```

Temporarily change the display format used for a date by using the AS DATE clause. The following example changes the display format of the current date obtained by the system variable @DATE:

```
OPEN order;
LIST ordernum,
     salesman,
     TITLE "ORDERS" SPACE 20,
     @DATE AS DATE "MA DB2,Y4" SKIP 2;
```

Report:

```
ORDERS                  MAY 7, 1982

ORDERNUM  SALESMAN
--------  --------

      21       205
      25       212
      30       222
     ...       ...
```

Change the display format of a date value that is not stored in internal format by including the JULIAN-DATE clause as follows:

```
OPEN order;
LIST ordernum,
     JULIAN-DATE((1900 + oyear),omonth,oday)
     AS DATE "DA - MA DB2,Y4",
     HEADING " DAY AND DATE ORDERED";
```

Report:

```
ORDERNUM        DAY AND DATE ORDERED
--------  --------------------------

      21     TUESDAY - JANUARY 10,1978
      25      MONDAY - JANUARY 23,1978
      30      MONDAY - FEBRUARY 6,1978
      32        FRIDAY - MARCH 17,1978
    ...            ...
```

PRINTING A TIME VALUE ON A REPORT. Use the AS TIME clause to specify printing of a time value in internal format on a report. The AS TIME clause prints the time value by using the default time format or a user-created time format.

Use the AS TIME clause to modify the system variable @TIME which returns the current time. The following example prints the time value with the default time format:

```
OPEN order;
LIST ordernum,
     JULIAN-DATE((1900 + oyear),omonth,oday)
     AS DATE "DA - MA DB2,Y4),
     HEADING " DAY AND DATE ORDERED",
     TITLE "OLD ORDERS" SPACE 10,
     @TIME AS TIME *;
```

Report:

```
OLD ORDERS          09:51:32 AM
ORDERNUM
--------

      21
      25
      30
      32
```

Use the AS TIME clause to print the time with a user created format as in the following example:

```
OPEN order;
LIST ordernum,
     TITLE "OLD ORDERS" SPACE 10,
     @TIME AS TIME "HPB2:M2";
```

Report:

```
OLD ORDERS          9:51 AM

ORDERNUM
--------
      21
      25
    ...
```

USING THE HELP COMMAND

ENFORM displays the syntax of statements, clauses, and commands when you enter the ?HELP command. Using the ?HELP command is particularly useful during an interactive session when you want to refresh your memory about how to specify a particular ENFORM language element. To use the ?HELP command, simply enter "?HELP." ENFORM responds by displaying a list of the elements for which help is available.

To obtain the syntax of a particular element on this list, enter the ?HELP command followed by the element name. For example, suppose you have forgotten the syntax used to specify the FIND statement. If you enter:

```
>?HELP FIND
```

ENFORM displays the following:

```
The FIND statement has the following form:

FIND [ UNIQUE ] <output-record-name>
     ( { [ <output-field-name> : = ] { BY <field name>
                                     { BY DESC <field name>
                                     { <target item>
                                     { ASCD <target item>
                                     { DESC <target item>
     } ,... )
    WHERE <logical expression> ;
```

Help is also available for TARGET ITEM and LOGICAL EXPRESSION

SECTION 4

COMPILING AND EXECUTING A QUERY

Before returning the information you want, ENFORM must compile and execute your query. ENFORM compiles and executes a query during a session (the period of time that begins when you enter the ENFORM command and ends when ENFORM terminates).

During a session, you can use ENFORM either in noninteractive mode or in interactive mode. The following paragraphs describe the use of ENFORM in both noninteractive and interactive mode.

USING ENFORM IN NONINTERACTIVE MODE

You can use ENFORM in noninteractive mode by specifying the ENFORM command with the IN option. For example:

```
:ENFORM/IN test1,OUT $s/
```

When you enter the ENFORM command in this form, ENFORM compiles (if necessary) and executes your program immediately. For the input file, you can specify either an Edit file containing ENFORM source code or a compiled query file (created previously by the ?COMPILE command). The input file must contain all the statements, clauses, and commands needed for the query.

Figure 4-1 shows a sample of the output produced when ENFORM is used in noninteractive mode.

```
ENFORM - T9102C09 - (02APR82)   DATE - TIME :  8/17/82 - 15:31:32
SOURCE FILE NAME IS $MKT.SAMPLE.TEST1


    1     * ?SOURCE addfile1
    2     * SET @display-count to 24;
    3     * OPEN employee;
    4     * LIST empnum, empname, job;
Employee
 Number        EMPNAME            JOB
--------  -----------------  ------------

       1  ROGER GREEN         MANAGER
      23  JERRY HOWARD        MANAGER
      29  JACK RAYMOND        MANAGER
     ...  ...                 ...
    4     * CLOSE employee;
    5     * OPEN parts,odetail,order;
    6     * LINK parts TO odetail VIA partnum;
    7     * LINK odetail TO order VIA ordernum;
    8     * LIST partname,deldate,quantity;
     PARTNAME       DELDATE   QUANTITY
-----------------  --------  --------

SYSTEM 192KB SEMI   04/10/78         1
DECIMAL ARITH       04/10/78         2
...                 ...             ...
** END-OF-ENFORM-RUN **
```

Figure 4-1. ENFORM in Noninteractive Mode

If you specify the name of an Edit file for the IN option of the ENFORM command, ENFORM compiles and executes all the statements and commands in the Edit file with the following exception: ENFORM ignores a ?RUN command in the Edit file.

If you specify the name of a compiled query file for the IN option of the ENFORM command, ENFORM executes all the statements and clauses in the compiled query file and any ?ASSIGN, ?DICTIONARY, or ?EXIT commands. Using a compiled query file as the input file is particularly useful because you can pass a parameter value to a compiled query file. For example, suppose you frequently issue the same query for different branches of your business. You could store several queries in Edit files with the only difference between the queries being the branch number. Alternatively, you could use the ?COMPILE command to compile the query and pass the branch number as a parameter before executing the compiled query file. For example, the following example shows the contents of the compiled query file *reportb*:

```
PARAM bnum I2;
TITLE "Employee Names and Salaries for Branch " (bnum);
OPEN employee;
LIST BY branchnum,
       empname,
       salary,
    WHERE branchnum = bnum;
```

To obtain a report for each branch, simply change the value you specify in the Command Interpreter PARAM command before you issue ENFORM command. For example:

```
:PARAM bnum 1
:ENFORM/ IN reportb, OUT $s/
:PARAM bnum 2
:ENFORM/ IN reportb, OUT $s/
...
```

When you use ENFORM in noninteractive mode, the session terminates as soon as an end-of-file is encountered on the input file.

USING ENFORM IN INTERACTIVE MODE

Use ENFORM in interactive mode by specifying the ENFORM command without the IN option. For example:

```
:ENFORM
```

ENFORM responds by displaying the ENFORM prompt (>). You can enter source code either directly or indirectly.

Entering Source Code Directly

Enter source code directly by typing ENFORM statements, clauses, and commands in response to the ENFORM prompt. End each statement with a semicolon so that ENFORM displays any error occurring during statement execution immediately. ENFORM compiles and executes your program as soon as it encounters the terminating semicolon of a LIST or FIND statement. For example, consider Figure 4-2.

```
:ENFORM

 ENFORM - T9102C08 - (02APR82)  DATE - TIME: 5/26/82 - 13:43:11

>OPEN parts;
>LIST parts;
   Part
 Number       PARTNAME        INVENTORY  LOCATION    PRICE
 ------  ------------------  ---------  --------  ---------

    212  SYSTEM 192KB CORE           7  J87        92000.00
    244  SYSTEM 192KB SEMI           3  B78        87000.00
```

Figure 4-2. Entering Statements Directly in Interactive Mode

If an error occurs during statement execution, use the FC command (a feature of all Tandem subsystems) to correct the last line entered or to reissue the previous command or statement. The FC command has three subcommands of its own:

- i insert one or more characters.
- r replace one or more characters.
- d delete one or more characters.

Cancel the FC command by typing two right slashes (//) in columns 1 and 2, by pressing the CTRL and Y terminal keys simultaneously, or by pressing the terminal BREAK key. Cancelling the FC command cancels the previous statement or command and returns you to the ENFORM prompt.

Refer to the *GUARDIAN Operating System Command Language and Utilities Manual* for more information about the FC command.

When entering source code directly, terminate the ENFORM session by issuing the EXIT statement or ?EXIT command, or by pressing the CTRL and Y terminal keys simultaneously.

Entering Source Code Indirectly

Enter source code indirectly by first placing it on an Edit file. Place any query that is longer than three or four lines or that is issued repeatedly on an Edit file.

Enter the EDIT Text Editor from the ENFORM subsystem by using the ?EDIT command. For example:

```
:ENFORM

ENFORM - T9102C09 - (02APR82)  DATE - TIME: 5/26/82 - 13:45:09

>?EDIT test1
TEXT EDITOR - T9601C08 - (01APR82)
CURRENT FILE IS $DATA.TEST.TEST1
*ADD 1
   1    OPEN parts;
   2    ...
```

Once you enter the Editor, all of its features are available.

Entering the ?EDIT command sets the default Edit file for subsequent ?RUN or ?EDIT commands. When you set the default Edit file, you need not include an Edit file name with a subsequent ?RUN or ?EDIT command when the same Edit file is to be used.

If you want to place more than one query on the same Edit file, you can identify a particular collection of statements or commands by using the ?SECTION command in the Edit file. For example:

```
:ENFORM

ENFORM - T9102C09 - (02AOR82)  DATE - TIME: 5/26/82 - 13:49:16

>?EDIT test2
TEXT EDITOR - T9601C08 - (01APR82)
CURRENT FILE IS $DATA.TEST.TEST2
*ADD 1
   1    ?SECTION testa
   2    OPEN parts;
   3     .
   4     .
   5     .
   6    ?SECTION testb
   7    OPEN employee;
   8     .
   9     .
```

Use the ?RUN command to compile and execute the source code contained on the Edit file. For example:

```
:ENFORM

 ENFORM - T9102C09 - (02AOR82)  DATE - TIME: 5/26/82 - 13:50:23

>?EDIT test3
TEXT EDITOR - T9601C08 - (01APR82)
CURRENT FILE IS $DATA.TEST.TEST3
*ADD 1
   1     OPEN parts;
   ...
*  EXIT
>?RUN
...
```

Notice that including an Edit file name with the ?RUN command is not necessary because the ?EDIT command sets the default Edit file.

Include a section name with the ?RUN command if you want to compile and execute only the source code in that section. For example:

```
:ENFORM

ENFORM - T9102C09 - (02APR82)  DATE - TIME: 5/26/82 - 14:30:15

>?EDIT test4
TEXT EDITOR - T9601C08 - (01APR82)
CURRENT FILE IS $DATA.TEST.TEST4
*ADD 1
   1   ?SECTION rpt1
               .
               .
               .
   45  ?SECTION rpt2
               .
               .
               .
*EXIT
>?RUN test4 (rpt2)
```

Specifying only the Edit file name in the ?RUN command causes ENFORM to compile and execute all the statements and commands in the Edit file even if ?SECTION commands are included.

If you frequently use the same statements and commands, place them in an Edit file and use the ?SOURCE command to read them into your ENFORM programs. For example:

```
:ENFORM

ENFORM - T9102C08 - (01DEC81)  DATE - TIME: 5/26/82 - 13:52:45

>?EDIT test5
TEXT EDITOR - T9601C08 - (01APR82)
CURRENT FILE IS $DATA.TEST.TEST5
*ADD 3
   1  ?SOURCE  addfile
   2  OPEN parts;
   3  LIST parts;
*EXIT
>?RUN
```

If you use the ?SOURCE command to read an Edit file containing a LIST or FIND statement, ENFORM compiles and executes the associated query as soon as the LIST or FIND statement is encountered.

Use the ?COMPILE command to compile a source program without executing it. The ?COMPILE command compiles the source code and saves the compiled query in a physical file. The ?COMPILE command compiles source code containing either LIST or FIND statements; however, only one LIST or FIND statement can be compiled in a source program. If more than one LIST or FIND statement exists in a source program, ENFORM compiles up to the first statement encountered and then executes any subsequent statements. You should compile source programs that: are called from a host language program, are passed parameters, or that you want to protect from inadvertent changes. In the following example the source program is created on *report1* and compiled and saved on the compiled query file *comprpt*:

```
:ENFORM

 ENFORM - T9102C09 - (02APR82)  DATE - TIME: 5/26/82 - 13:59:05

>?EDIT report1
TEXT EDITOR - T9601C08 - (01APR82)
CURRENT FILE IS $DATA.TEST.REPORT1

* ...
*EXIT
>?COMPILE report1 TO comprpt
```

You can pass a value to a parameter in a compiled query file when you are using ENFORM in interactive mode. To pass a parameter value, issue the Command Interpreter PARAM command before entering the ENFORM command. For example:

```
:PARAM rptnum 3
:ENFORM

 ENFORM - T9102C09 - (02APR82)  DATE - TIME: 5/26/82 - 13:59:05
```

You can then use the ?EXECUTE command to execute the stored compiled query file.

When entering source code indirectly, terminate an ENFORM session by entering either the EXIT statement or ?EXIT command or by pressing the CTRL and Y terminal keys simultaneously.

SECTION 5

USING ENFORM EFFICIENTLY

After you develop a successful ENFORM query, consider optimizing the efficiency of the query. This section provides information about using the ENFORM statistics to examine the efficiency of your query and about changes that you can make to improve query performance.

USING ENFORM SEARCH STATISTICS

When you set the Option Variable @STATS to ON, ENFORM produces search statistics. An understanding of these statistics is useful when you are attempting to improve the performance of your ENFORM queries.

Figure 5-1 shows a sample query and the resulting statistics. The following paragraphs explain these statistics.

```
SET @STATS TO ON;
OPEN parts,odetail,order;
LINK parts TO odetail VIA partnum;
LINK order TO odetail VIA ordernum;
LIST BY partname,BY order.ordernum.quantity;

        FILE NAME                LEVEL READ    RECORDS READ    POSITIONS
$MKT.SAMPLE.parts                     3             3              3
$MKT.SAMPLE.order                     2             3             127
MKT.SAMPLE.odetail                    1           127               1
003,08,052          BEGIN(8/16/82 — 09:03:02:41)      END(8/16/82 — 09:03:14:78)
      STRATEGY COST = 2
```

Figure 5-1. Simple ENFORM Query and Associated Search Statistics

The FILE NAME Column

The first column, headed FILE NAME, provides the names of the different physical files that the query processor reads to obtain the retrieved data. In Figure 5-1, the query processor read the physical files named *$mkt.sample.parts*, *$mkt.sample.order*, and *$mkt.sample.odetail*.

The LEVEL READ Column

The second column, headed LEVEL READ, provides a number that identifies the sequence in which the query processor read the physical files. The first file read has a value of 1, the second file read has a value of 2, and so on. In Figure 5-1, the first physical file read is *$mkt.sample.odetail*, the second physical file read is *$mkt.sample.order*, and the third physical file read is *$mkt.sample.parts*.

Figure 5-2 shows sample search statistics where both *$mkt.sample.region* and *$mkt.sample.employee* indicate LEVEL READ as 1.

```
SET @STATS TO ON;
OPEN region,employee;
LINK region TO employee VIA regnum;
LIST BY regname,empname;

        FILE NAME          LEVEL READ   RECORDS READ   POSITIONS

$mkt.sample.region              1             7             1
$mkt.sample.employee            1            55             1
003,06,022 BEGIN( 8/31/82 - 15:17:15:57)        END ( 8/31/82 - 15:17:27:33)
        STRATEGY COST = 3
```

Figure 5-2. Search Statistics where LEVEL READ = 1

The search statistics shown in Figure 5-2 occur because the query processor sets up a special join strategy (a means of associating data from the two files) to search the two files at the lowest level. In order for the query processor to set up a join strategy, the two files must be sorted by the value of the linking fields. To retrieve the records requested in the query shown in Figure 5-2, the query processor reads the first record from *$mkt.sample.region*. The query processor then reads the first record from *$mkt.sample.employee*. If this record matches the record from *$mkt.sample.region*, the query processor builds a target record from the matching pair. The query processor continues reading records in *$mkt.sample.employee* until the value of the linking field in *$mkt.sample.employee* exceeds the value of the linking field in *$mkt.sample.region*.

The query processor then reads the next record from *$mkt.sample.region*. If the linking field in *$mkt.sample.region* is the same for the newly read record as for the previously read record, the query processor rereads the records from *$mkt.sample.employee* for which the linking fields match. For the query shown in Figure 5-2, no duplicate values exist for the linking field in *$mkt.sample.region* because the linking field is the primary key of that file. The query processor, therefore, reads *$mkt.sample.employee* until the value of the linking field exceeds the value of the linking field in the current record from *$mkt.sample.region*.

The query processor continues reading the two files in parallel until both files have been read completely. A query that causes the query processor to use the join strategy is desirable because each file is read only once.

The RECORDS READ Column

The third column, headed RECORDS READ, provides a number that indicates the number of records that the query processor read in each physical file. The number indicates logical reads. Logical reads (as opposed to physical reads) count the requests for records but ignore reads of alternate key data files and index blocks. Logical reads also ignore the effect of buffering, which might reduce physical reads. For example in the query shown in Figure 5-1, the query processor logically read a record from *mkt.sample.odetail* 127 times. Normally, several times that number of physical reads occur.

The presence of keys can affect the number of reads performed. If the file read second has an alternate or primary key field by which it is linked to the file read first, the query processor positions on that key and then reads only the appropriate subset of records. If a linking key field does not exist in the file read second, the query processor must read the entire file for each record selected in the file read first.

The POSITIONS Column

The fourth column, headed POSITIONS, provides a number that indicates the total number of positions that the query processor performs on each physical file. Note the relationship between the number of records read in the first file and the number of positions performed on the second and third files. The statistics in Figure 5-1 indicate that to retrieve the information request in the query the query processor:

1. Positioned *$mkt.sample.odetail* to beginning of information.

2. Read from *$mkt.sample.odetail.*

3. Positioned *$mkt.sample.order* according to the primary key *ordernum.*

4. Read from *$mkt.sample.order.*

5. Positioned *$mkt.sample.parts* according to the primary key *partnum.*

6. Read from *$mkt.sample.parts.*

The query processor repeated steps 2 through 6 until it retrieved the requested information. The query processor retrieved the information in this manner because both *ordernum* and *partnum* are the primary key of their respective files.

The Identification Line

The line below the four columns of reported statistics identifies the ENFORM process (network node number, CPU number, and process number), the date and time the query was initiated, and the date and time the query terminated.

The STRATEGY COST Line

The final line of the statistics contains the value of STRATEGY COST. Possible values of STRATEGY COST for successfully executed queries range from 1 to 8. In Figure 5-1, the strategy cost is 2. The significance of the strategy cost values is as follows:

- STRATEGY COST = 1

  The query processor used keyed access on all files. A strategy cost of 1 might occur when a single file is involved in a query with a request qualification (in a WHERE clause) that compares the primary key of the file to a constant. This type of query forces the query processor to access the records by the primary key.

- STRATEGY COST = 2

  The query processor might have to read one file completely. A strategy cost of 2 might occur when a single file is involved in a query with a request qualification that specifies a nonkey field, compares a key field to a constant using the NOT EQUAL operator, or compares a key field to another field in the same record.

- STRATEGY COST = 3

  The query processor might have to read more than one file completely. A strategy cost of 3 might occur when two files are involved in the query and the query processor uses the join strategy described previously in this section.

- STRATEGY COST = 4

  The query processor must sort one file once for a join strategy. A strategy cost of 4 might occur when two files are involved in a query and one of the fields used to link record descriptions is either a nonkey field or a partial key field.

- STRATEGY COST = 5

  The query processor must sort more than one file once for a join strategy. A strategy cost of 5 might occur when two files are involved in a query and both of the fields used to link record descriptions are either nonkey fields or partial key fields.

- STRATEGY COST = 6

  The query processor might have to sort one file more than once for a link. A strategy cost of 6 might occur when:

  A query involves at least three files.

  The query processor uses a join strategy on the last two files read.

  The query contains a request qualification that compares fields from one of the files involved in the join strategy with fields from a nonjoin file.

  The query processor must sort one of the files involved in the join strategy.

- STRATEGY COST = 7

  The query processor might have to sort two files more than once for a link. A strategy cost of 7 might occur when:

  A query involves at least three files.

  The query processor uses a join strategy on the last two files read.

  The query contains a request qualification that compares fields from both of the files involved in the join strategy with fields from a nonjoin file.

  The query processor must sort both files involved in the join strategy.

- STRATEGY COST = 8

  The query processor reads more than one file completely at least once with no convenient strategy for linking the record descriptions. A strategy cost of 8 might occur when two or more files are linked with non-equality comparisons in a request qualification. A strategy cost of 8 frequently occurs when a query links two or more FIND files in which the data is stored in the default unstructured file type.

IMPROVING PERFORMANCE

The following paragraphs suggest changes that improve the performance of ENFORM queries. Consider making changes in three areas:

- The data (disc) environment
- The nondisc environment
- The wording of the query itself

Only make changes that: are easy to implement, generate significant reduction in query response time, and increase transaction rate (the updating and retrieval of data) while reducing response time.

Several of the suggested changes are helpful in all circumstances. Others are helpful only for particular cases. Use your knowledge of your own environment to determine the changes most helpful to you.

Changing the Data (Disc) Environment

The most desirable changes to the disc environment both decrease response time and increase the transaction rate. These changes involve reducing the number of physical file accesses by:

- Removing levels of indexing in key-sequenced files.
- Adding or removing alternate keys.
- Avoiding sorting of an already sorted file.
- Specifying where ENFORM builds temporary work files.
- Spreading input/output demands among discs.
- Altering cache size.
- Controlling the size of the target file.

REMOVE LEVELS OF INDEXING IN KEY-SEQUENCED FILES. Reducing levels of indexing in key-sequenced files decreases the number of disc input-output (i/o) operations needed. Note that any alternate key file is a key-sequenced file.

To remove levels of indexing, increase the index block size (allowing enough slack for anticipated growth) and reload the data. Increasing the index block size might reduce the number of index levels and therefore, both reduce the number of physical accesses to the disc and improve response time.

Key-sequenced files with small block sizes usually have more levels of index blocks because small blocks fill faster than large blocks. In particular, key-sequenced files created by the FUP output of the Data Definition Language (DDL) have small blocks because they use the 1024-byte block size provided by DDL. Consider Figure 5-3 which shows a diagram of a key-sequenced file with a 1024-byte block size.

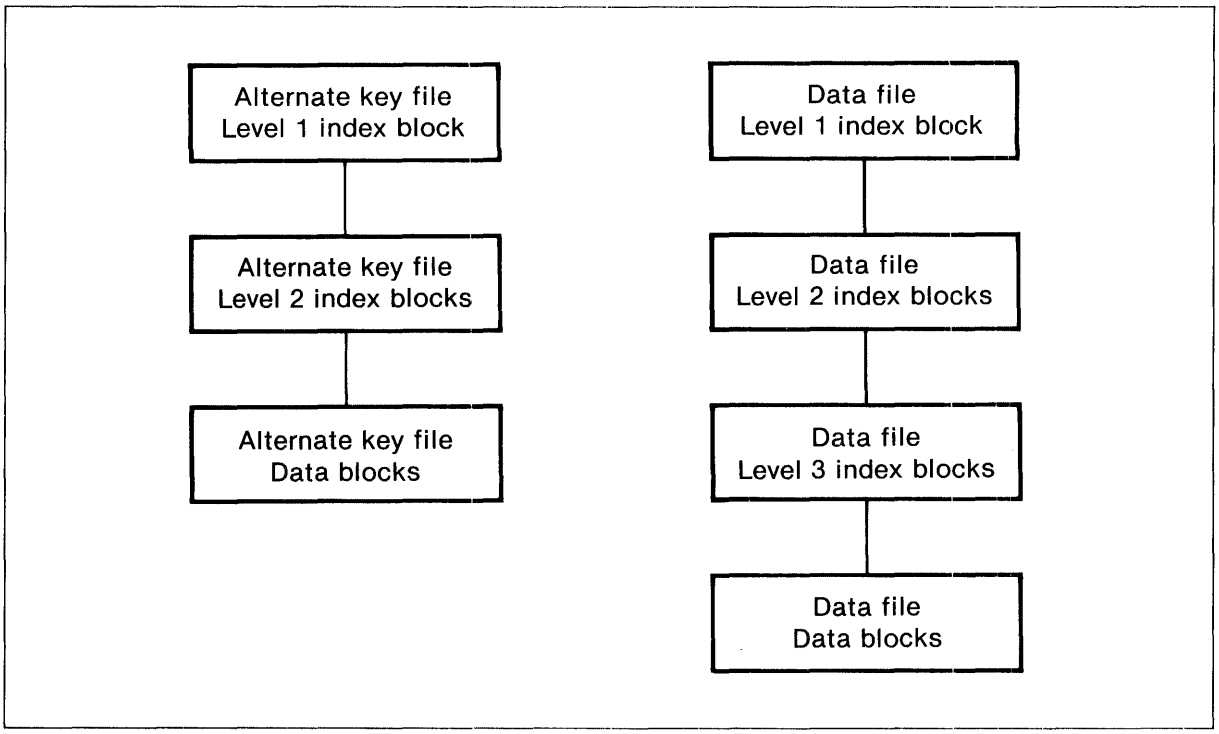


Figure 5-3. Diagram of Key-Sequenced File with 1024-Byte Block Size

In Figure 5-3, the alternate key file has two levels of index blocks plus the data block. The data file has three levels of index blocks plus the data block. When using the alternate key search path to retrieve a record from this file, the query processor must access seven blocks: three alternate key blocks and four data file blocks. Even if the query processor has accessed the files recently so that both level 1 index blocks are in cache, it must still access five blocks: a level 2 alternate key file index block, an alternate key file data block, a level 2 and a level 3 data file index block, and a data file data block.

Figure 5-4 shows the same key-sequenced file after the block size is increased to 4096 bytes.

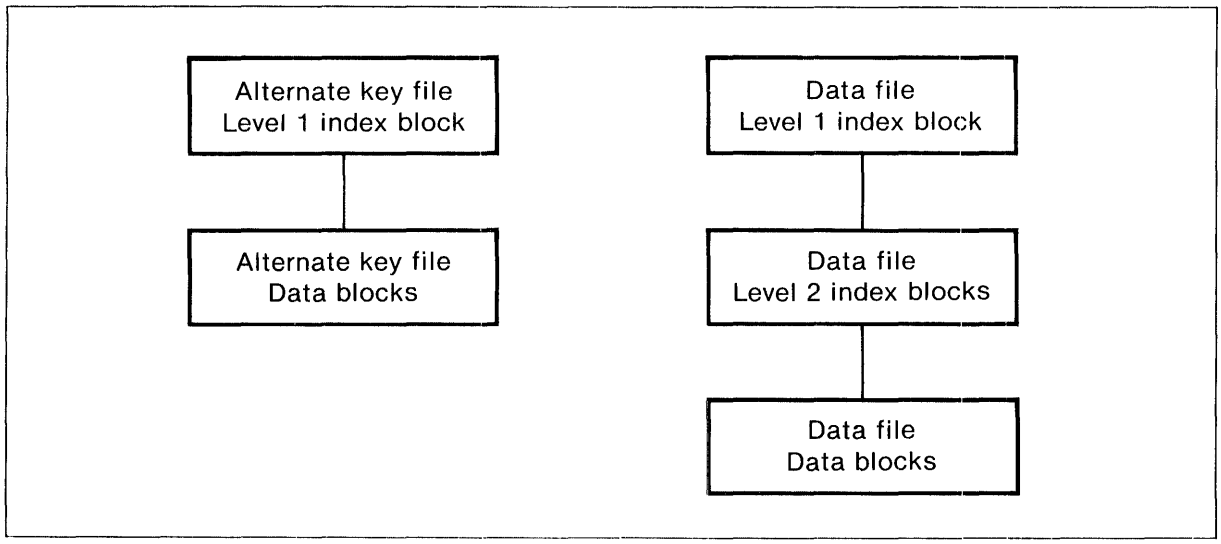


Figure 5-4. Key-Sequenced File with Increased Block Size

Increasing the block size eliminates one index block for both the alternate key file and the data file. After the data is reloaded, the query processor must access only five blocks to retrieve the required record: two blocks in the alternate key file and three blocks in the data file. If both level 1 index blocks are in cache, the query processor must access only three blocks: an alternate key data block, a level 2 data file index block, and a data file data block.

For the file in Figure 5-4 increasing the block size has decreased the number of physical file accesses from 7 to 5. This reduction in physical file accesses generates a significant improvement in query performance.

Note that the effect of increasing the block size is not apparent in the ENFORM statistics. ENFORM reports one logical data access in its statistics no matter how many physical accesses occur.

ADD OR REMOVE ALTERNATE KEYS. Adding or removing alternate keys improves performance if a reduction in the number of physical file accesses results. The conditions under which adding or removing alternate keys affect performance are:

- When the alternate key field is used to select a small subset of records, adding an alternate key reduces the number of physical file accesses.

- When the alternate key field is used to select a large subset of records, removing the alternate key reduces the number of physical file accesses. It is more efficient for the query processor to sort the file or simply access it sequentially when the number of records retrieved is large.

Note that the addition or removal of alternate keys affects other transactions that access the physical file. For example, PATHWAY often uses alternate keys to access a tiny subset of the data base. Thus, removal of an alternate key might improve the performance of ENFORM but have a disastrous effect on a PATHWAY application. Consider the cost of record insertions as a consequence of adding alternate keys. The cost of record insertions is an important consideration when determining whether to add an alternate key since it could significantly slow application response time and transaction rate.

Adding Alternate Keys. To observe the effect of adding an alternate key that is used to select a small subset of records, consider *$mkt.sample.employee*, the physical file that stores the records for *employee* (described in Appendix C). Assume that no alternate key exists for *employee* and that *$mkt.sample.employee*:

- Stores one hundred records.
- Has ten data blocks each containing ten records.
- Has one index block.

When the following query is issued:

```
OPEN employee;
LIST empnum, empname, salary,
  WHERE empname = "TOM HALL";
```

the query processor must access the index block and all ten data blocks of *$mkt.sample.employee* to retrieve the required record, resulting in a total of eleven physical file accesses.

Consider the effect of making *empname* an alternate key for *employee*, such that the alternate key file:

- Stores one hundred records (one for each data record).
- Has ten data blocks each containing ten records.
- Has one index block.

If the preceding query is now issued, the query processor must access the alternate key file index block, one alternate key file data block, one *$mkt.sample.employee* index block, and one *$mkt.sample.employee* data block, resulting in a total of four physical file accesses.

The reduction in physical file accesses is over 60%. The principles described in this example apply to files of any size. For very large files, the resulting reduction in physical file accesses is very significant.

Removing Alternate Keys. To observe the effect of removing an alternate key that is used to select a large subset of records, assume that *$mkt.sample.employee* has an alternate key field called *job*, and the alternate key file:

- Stores one hundred records (one for each data record).
- Has ten data blocks each with ten records.
- Has one index block.

Assume also that seventy records in $mkt.sample.employee contain the value SALESMAN for the *job* field. When the following query is issued:

```
OPEN employee;
LIST empnum, empname, job,
  WHERE job EQ "SALESMAN";
```

the query processor must access the alternate key index block, one alternate key data block, the *$mkt.sample.employee* index block, and the *$mkt.sample.employee* data block to obtain the first record. Assume that both index blocks remain in cache. The query processor must access approximately 8 of the alternate key data blocks. For each of the remaining 69 records, it is unlikely that the necessary data file data block is in cache so the query processor must access the *$mkt.sample.employee* data blocks 69 different times to obtain the remaining records. In this case the presence of an alternate key results in a total of 80 physical file accesses.

If *job* is removed as an alternate key and the preceding query is issued, the query processor reads *$mkt.sample.employee* sequentially, accessing only its index block and 10 data blocks. Removing the alternate key results in a total of 11 physical file accesses.

The physical file accesses decrease by 88% when the alternate key is removed. The reduction in physical file accesses will be proportionally similar when a file with a large number of records is involved.

AVOID SORTING AN ALREADY SORTED FILE. Avoid the processing time ENFORM spends sorting an already sorted file by including the SEQUENCE IS clause in the dictionary record description. Of course, the processing time is eliminated only when your query specifies a sorting order that matches the order specified in the SEQUENCE IS clause. The time eliminated depends on your query and the size of the file.

SPECIFY WHERE ENFORM BUILDS TEMPORARY WORK FILES. During an ENFORM session the query processor and the SORT process build temporary work files while processing a FIND or LIST statement. Use generic files to specify where these temporary files are built. Directing the query processor temporary work files to a scratch disc is particularly useful. For example:

```
:ASSIGN QUERY-WORK-AREA, $SLOW
```

Place temporary files built by the SORT process on less used volumes. For example:

```
?ASSIGN QUERY-SORT-AREA, $QUIET
```

SPREAD INPUT/OUTPUT DEMANDS AMONG DISCS. Distribute data among different disc volumes to reduce the demands on any one disc process. Distribute data by:

- Partitioning files, especially those that are large and heavily used.
- Moving data files to less used volumes.
- Placing alternate key files on a different volume than the associated data file.

The improvement shown depends upon the organization of your system.

ALTER CACHE SIZE. The value of cache size is very application dependent. If main memory space is not needed for other purposes, it can be valuable to make cache large enough to hold every top level index block of each heavily accessed file, with a little space left over. In practice, a reasonable starting allotment for cache is 30 pages per CPU containing disc processes or 10 pages per disc process on a Tandem Nonstop II system.

Remember that cache uses main memory. The memory used for cache is not available for other processes. When too much memory is used for cache, page faults result. As cache size increases, the time needed to manage cache also increases. In a system where files compete for cache (even very large cache), increasing cache size might not improve performance.

CONTROL THE SIZE OF THE TARGET FILE. Control the primary and secondary extent sizes for the target file. If extents are too large, disc space is wasted and the query might fail because an extent cannot be allocated. If extents are too small, the target file might overflow the allocated space. When the target file overflows, the query processor creates a new larger target file and copies the old target file into the new one. This process of creating a new target file and copying the old one degrades query performance, especially if the query processor must do it more than once.

By default, the approximate extent sizes are determined as follows:

- For the primary extent size of the target file, ENFORM uses whichever is larger:

  the size (in bytes) of the largest input file divided by 2048

  the largest primary extent size of all the input files.

- For the secondary extent size of the target file, ENFORM uses whichever is larger:

  the largest secondary extent size of all the input files

  the size determined for the primary extent of the target file

The default extent sizes used by ENFORM are satisfactory in many cases. In other cases, however, especially if the target file or one of the input files is very large, the default extent sizes are either too small or too large.

Fortunately, you can control the extent sizes. One way to control the extent sizes is to estimate the number of target records and set the @TARGET-RECORDS Option Variable clause to that number. ENFORM, then, uses the number specified to determine the primary and secondary extent sizes for the target file. When @TARGET-RECORDS is specified, ENFORM uses the following formula to determine the size of both the primary and secondary extents:

(number of target records * target-record-length)/20480

The most accurate way to control the primary and secondary extent sizes of the target file is to set both the @PRIMARY-EXTENT-SIZE and @SECONDARY-EXTENT-SIZE Option Variable clauses appropriately.

Changing the Nondisc Environment

Desirable changes to the nondisc environment reduce the number of processes competing for a CPU, the number of processes competing for a system, and the number of processes competing for a network. Possible changes are:

- Placing the compiler/report writer process and a server query processor in different CPUs.
- Sharing query processors.
- Reducing network traffic.

PROCESS PLACEMENT. To reduce the number of processes competing for a CPU, start a server query processor in a different CPU than the one that contains the disc process for the files to be accessed. If possible start the compiler/report writer process in a different CPU than the one containing either the server query processor or the disc process.

Starting a server query processor in a different CPU than the one containing the compiler/report writer is useful in reducing problems with shortpool (an area of shared memory) on the Tandem NonStop System.

Remember to use the ?ATTACH command to identify the server query processor.

SHARE QUERY PROCESSORS. To reduce the number of processes competing for a system, create one or more shareable server query processors. The major advantages of a shareable server query processor are:

- Generally fewer processes exist within the system which results in reduced use of system resources such as memory paging and disc arm movement.
- The number of open operations is reduced because the Command Interpreter ASSIGN command can be issued to hold files open for the life of the server query processor.

Each shared server query processor can have a different purpose. For example, create different shared server query processors for:

- Short queries where a response is expected in a few seconds.
- More lengthy queries that require more processing time.
- Limited access files. In this case the logon default of the person creating the query processor determines who has access to the files.
- General access for those who need it.

REDUCE NETWORK TRAFFIC. When your process searches data on a remote system, reduce competing processes by starting ENFORM on that system and using the spooler to transfer the data. Using spooler-to-spooler communication takes maximum advantage of the network bandwidth. An additional advantage is that you can hold the records retrieved in the spooler for later printing or for printing if the network link terminates.

Alternatively, either start a server query processor on the remote system or use a server query processor that already exists on that system to obtain the data. In this case, the server query processor transmits the data to a query compiler/report writer process on your system.

Changing the Wording of the Query Itself

Changing the wording of a query often leads to performance improvement because of the way in which ENFORM determines its search strategy. ENFORM analyzes each query to determine the optimal search strategy. ENFORM attempts to minimize the number of physical file accesses needed by defining the sequence in which the physical files are accessed and by defining how each physical file is accessed (for example, by key value). ENFORM defines its file access sequence on the basis of information obtained from the dictionary record description and any qualifications for record selection specified in the query. This information is often incomplete.

To identify where possible changes in query wording might be helpful, examine the ENFORM statistics (described in this section). Determine the sequence in which the files are searched, the number of reads performed, and the number of positionings needed.

Some changes to query wording that might reduce physical file accesses are:

- Reduce the number of records selected by adding request qualifications (a WHERE clause) to the query.
- Experiment by changing the way field names are qualified in a WHERE clause.
- Determine if queries can share FIND files.

ADD A WHERE CLAUSE. Take advantage of information that is known to you but not known to the query processor and add a WHERE clause to your query. Adding a WHERE clause might force the query processor to search the data base in a more efficient manner by reducing the number of logical reads needed. Use the ENFORM statistics described earlier in this section to see if you are reducing the number of logical reads.

An understanding of the method that the query processor uses to examine the logical expression of a WHERE clause is useful. When the query processor determines its search strategy, it looks at the compiled query representation for the clause that most restricts the records in a particular file. The query processor searches the clauses looking for a logical expression where a primary key is equal to a constant. Failing to find such an expression, the query processor looks for a logical expression where an alternate key is equal to a constant. If neither exists, the query processor looks for a logical expression where a primary key is compared to a constant by a conditional operator such as less than or greater than. Finally the query processor looks for a logical expression where an alternate key is compared to a constant by a conditional operator.

For example, to force the query processor to search the *region* records first without restricting the number of target records returned, add a request qualification such as:

```
WHERE region.regnum GT 0;
```

When comparing a key to a constant, avoid specifying the key in an arithmetic expression such as:

```
WHERE key - 1 = x
```

Instead, specify the key alone on one side of the conditional operator and perform any arithmetic operations on the field or constant to which the key is compared:

```
WHERE key = x + 1
```

CHANGE THE QUALIFICATION OF FIELD NAMES IN A WHERE CLAUSE. Experiment with the way you qualify a field name in a WHERE clause if that same field has been specified in a LINK statement. When a linking field name is specified in a WHERE clause in the same query, qualify the field name with the record name that corresponds to the smallest number of records. The result remains the same no matter which record name is used to qualify the field name; however, qualifying a field name with the record name associated with the fewest records often results in a more efficient search strategy.

DETERMINE IF FIND FILES CAN BE SHARED. Analyze the queries made to determine whether they are sufficiently similar to share the same FIND (intermediate) files. Sharing FIND files improves performance because the query processor must select the data used for many reports only once. Sharing FIND files is particularly useful when the STRATEGY COST (see "Using the ENFORM Statistics" in this section) of the query is high and the intermediate file is small relative to the size of the orginal files.

Consider using FUP to load frequently used FIND file data into a structured file to also improve performance.

SECTION 6

HOST LANGUAGE INTERFACE

The ENFORM interface procedures provide you with the ability to use a host programming language as a high-level access facility to a relational data base. The ENFORM interface procedures communicate with the query processor.

By interfacing with ENFORM through the host programming language input/output interfaces, a host language program retrieves records at a fraction of the programming effort required by direct access. In addition if the file types, access keys, and logical design of the data base change, the host language program does not need modification because ENFORM obtains this information from the data dictionary.

When you write a host program in Tandem COBOL, FORTRAN, or TAL (Transaction Application Language), the host program interfaces with the query processor through a precompiled query. The compiled query must contain one and only one FIND statement. The record description for the new output file must be added to the dictionary before the query is compiled. These steps are shown in (1) and (2) of Figure 6-1.

The host language interface works as shown in (3) of Figure 6-1. In the host program you specify parameters and the name of the compiled query file to the query processor. The query processor retrieves the information you specified in the compiled query and returns records to the host program, one at a time.

INTERFACE PROCEDURES

A host language program interfaces with ENFORM by calling the three procedures: ENFORMSTART, ENFORMRECEIVE, and ENFORMFINISH. ENFORMSTART initiates the query processor, ENFORMRECEIVE provides records to the host program one at a time, and ENFORMFINISH terminates the query processor.

Each host application program interfacing with ENFORM must use: ENFORMSTART once for each compiled query; ENFORMRECEIVE once for each record to be provided to the host program (this is usually inside a loop); ENFORMFINISH once for each time you complete processing with a particular query processor; and the necessary global and local declarations.

A detailed description of each of the ENFORM procedures follows.

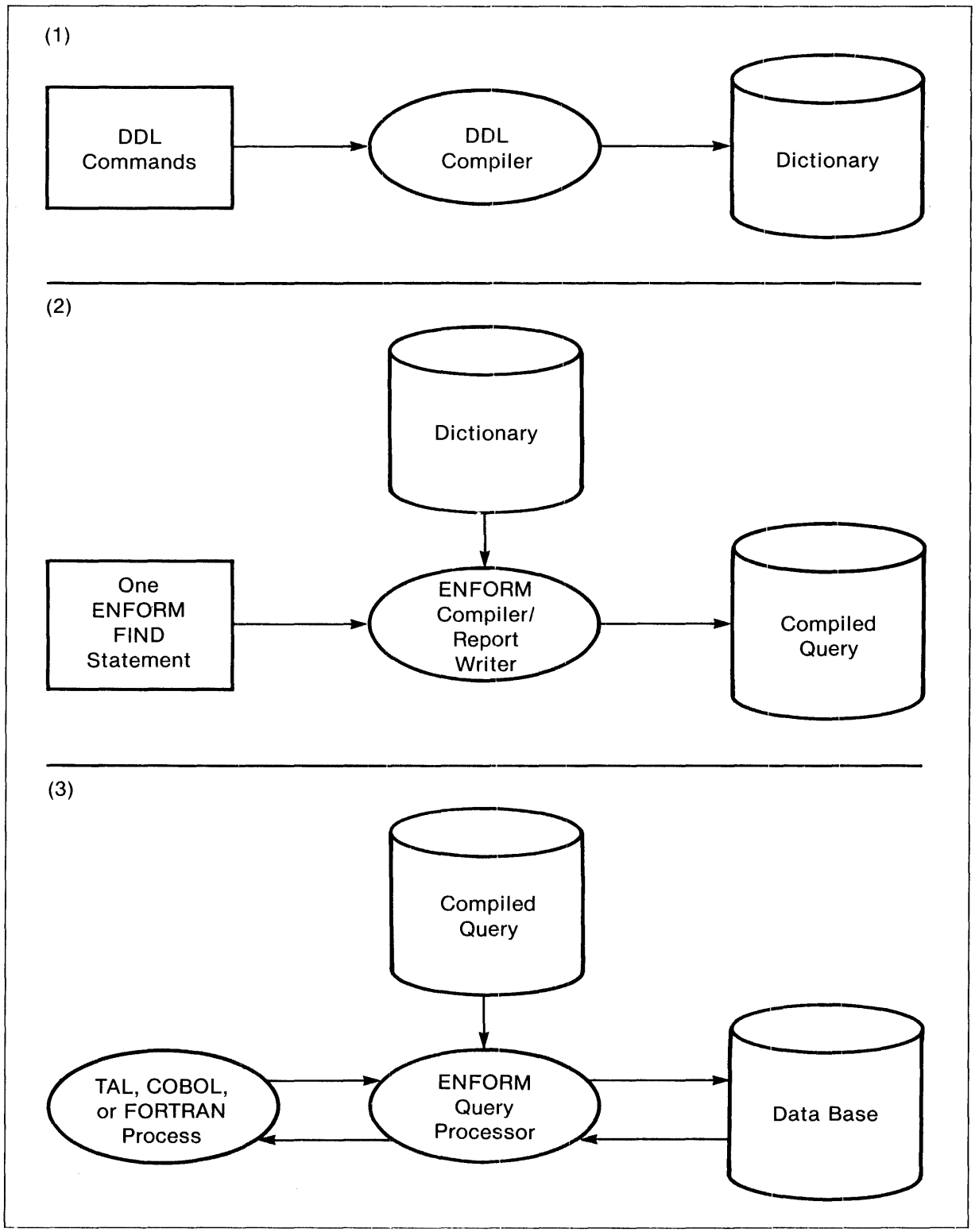


Figure 6-1. ENFORM Interface With Host Program

ENFORMSTART Procedure

The ENFORMSTART procedure initiates the interface of a host program with ENFORM. The syntax of the ENFORMSTART procedure is:

COBOL:

```
ENTER ENFORMSTART USING  ctlblock
                       , compiled-physical-filename
                       , buffer-length
                       , error-number
                       , [ restart-flag ]
                       , [ param-list ]
                       , [ assign-list ]
                       , [ process-name ]
                       , [ cpu ]
                       , [ priority ]
                       , [ timeout ]
                       , [ reserved-for-expansion ]
```

FORTRAN:

```
CALL ENFORMSTART (ctlblock
                 ,compiled-physical-filename
                 ,\buffer-length\ ,error-number
                 ,[ \restart-flag\]
                 ,[ param-list ]
                 ,[ assign-list ]
                 ,[ process-name ]
                 ,[ \cpu\ ]
                 ,[ \priority\ ]
                 ,[ \timeout\ ]
                 ,[ reserved-for-expansion ]
                 ,\maskword\ )
```

TAL:

```
CALL ENFORMSTART ( ctlblock
                 , compiled-physical-filename
                 , buffer-length
                 , error-number
                 , [ restart-flag ]
                 , [ param-list ]
                 , [ assign-list ]
                 , [ process-name ]
                 , [ cpu ]
                 , [ priority ]
                 , [ timeout ]
                 , [ reserved-for-expansion ])
```

→

where

`ctlblock`

INT:ref, is an 18-word integer array control block which must be supplied for global storage across the ENFORM procedure calls. This same storage is used in ENFORMRECEIVE and ENFORMFINISH, and any subsequent calls to ENFORMSTART. The host application program must not change this control block between calls to ENFORM.

`compiled-physical-filename`

INT:ref, is a 12-word array that specifies the physical file containing the compiled query. The file name must be specified as 24 characters in length with the:

$volume name	specified as 8 characters (blank filled if necessary)
subvolume name	specified as 8 characters (blank filled if necessary)
disc file name	specified as 8 characters (blank filled if necessary)

Refer to the *Guardian Operating System Programming Manual* for the exact form of a Tandem internal filename.

`buffer-length`

INT:value, is the length, in bytes, of the buffer that the process will use to receive records via ENFORMRECEIVE. *Buffer-length* must be at least 6. If information in addition to the error number is desired, *buffer-length* must be at least 30. See Table 6-3 later in this section.

`error-number`

INT:ref, is assigned an error number if an error condition occurs in ENFORMSTART or ENFORMRECEIVE. It is initialized to zero. *Error-number* should be declared global to the ENFORM procedures so that it can be checked after the ENFORMSTART or ENFORMRECEIVE procedures return. The error messages are described later in this section.

`restart-flag`

INT:value, is used when there is more than one query to be run by the host application program. A nonzero value causes the existing query processor to begin the next query with a new parameter and assign list, which is more economical than creating a new query processor for each query. When the existing query processor is used to begin the next query, the parameter values for *ctlblock*, *process-name*, *cpu*, and *priority* for the ENFORMSTART procedure must be identical to those for the initial call to the ENFORMSTART procedure or they must not be used.

A zero value for *restart-flag* causes a new query processor to be created. It is more expensive to start up a new query processor for each query run. →

param-list

INT:ref, is a pointer to the parameter (name:value) list in a form equivalent to the PARAM message generated by the Command Interpreter. See the *GUARDIAN Operating System Command Language and Utilities Manual* for the correct format.

assign-list

INT:ref, is a pointer to a sequence of one or more ASSIGN messages, each formatted as an ASSIGN message by the Command Interpreter. See the *GUARDIAN Operating System Command Language and Utilities Manual* for the correct format. These messages are preceded by a one-word header that contains a number equal to the total messages (31 or fewer messages) in the list.

Assign-list is used to override the physical filename or names for input files to the query. Any physical filename not expanded in the ASSIGN message will use the default volume and subvolume. If that is undesirable, the host language program must fill in the correct volume or subvolume after receiving ASSIGN messages from the Command Interpreter and before passing *assign-list* to ENFORMSTART. When *assign-list* is not used, the Tandem physical filename compiled into the query is used.

Assign-list can contain generic file names. Refer to the *ENFORM Reference Manual* for information about generic files. If the generic file QUERY-QPSTATUS-MESSAGE is specified from a host language program, ENFORM uses the message text from $SYSTEM.SYSTEM.ENFORMMK. There is no way to specify a new message table from a host language program.

The physical filename and its exclusion specification (Shared, Protected, or Exclusive) can be overridden. When not overridden, the query processor always opens the physical input files with shared mode.

process-name

INT:ref, is a four-word array that, if present, specifies the process name of a server query processor to use. If a query processor by that name does not exist, or if it cannot accept the query because the query processor is busy or the query exceeded its processing limits, an error results. If this parameter is omitted, a dedicated query processor for the query is created by ENFORMSTART and deleted by ENFORMFINISH. Refer to the *ENFORM Reference Manual* for information about a server query processor.

cpu

INT:value, selects the CPU in which to run a dedicated query processor. If omitted, the query processor runs in the same CPU as the host application program. When *process-name* is given, this parameter is ignored.

priority

INT:value, assigns a priority for a dedicated query processor. The default is the priority assigned the host application program. When *process-name* is given, this parameter is ignored. →

`timeout`

INT(32):value, if present, indicates the maximum time (in .01 second units) that the application process is willing to wait (i.e. be suspended) for the query processor to begin processing the query.

Timeout is most meaningful to use in the case of a request to a server query processor named with *process-name* that might be busy processing another query at the time ENFORMSTART is called. If ENFORMSTART returns an error 22, indicating that timeout has occurred, the application process could revert to a dedicated query processor by calling ENFORMSTART again without the process-name parameter.

If *timeout* is omitted, then the application process waits indefinitely for the query processor to begin the query.

`reserved-for-expansion`

INT:ref, reserved for expansion.

`mask-word`

INT:value, is a parameter that must be included when FORTRAN is used to call ENFORMSTART. Refer to the *FORTRAN 77 Reference Manual* for more information about this parameter.

Condition code settings:

< (CCL) means there is a problem. An error number representing the reason will be found in *error-number*.

= (CCE) means the query processor has been successfully initialized.

FORTRAN programs must contain special code in order to examine the condition code settings.

ENFORMSTART ERROR MESSAGES. If an error occurs during the execution of the ENFORMSTART Procedure, the number of the error is returned to *error-number*. Any of these error conditions terminate the query processor. If the query processor is dedicated, it is deleted. Table 6-1 lists the error message for each of the possible error numbers.

Table 6-1. ENFORMSTART Error Messages

Number	Message
3	An error occurred in trying to read the compiled-physical-filename.
8	An error occurred in trying to open the compiled-physical-filename.
15	A required ENFORMSTART parameter is missing.
16	Some failure occurred in creating or communicating with a query processor.
17	Either the param-list or assign-list is not in the correct format.
18	The value specified for buffer-length is less than 6.
19	A restart (restart-flag not equal to zero) call made to ENFORMSTART was done under the following invalid conditions: • No previous successful call to ENFORMSTART • ENFORMFINISH called before this ENFORMSTART • Last ENFORMRECEIVE did not end in end-of-file or error status
20	The file named by compiled-physical-filename cannot be executed. Either the physical file has an invalid file code for a compiled query, or the compiled query contains a LIST statement rather than a FIND statement.
21	The file named by compiled-physical-filename has an outdated version number.
22	The amount of time specified by the timeout parameter has elapsed with no response from the query processor.
23	Ctlblock was modified by the host application program since the last call to ENFORM procedures.
24	The amount of stack space needed to build the message to be sent to the query processor is not available.

ENFORMRECEIVE Procedure

The ENFORMRECEIVE procedure provides records to the host application program, one at a time. The syntax of the ENFORMRECEIVE procedure is:

COBOL:

```
ENTER ENFORMRECEIVE USING ctlblock, buffer [ GIVING count ]
```

FORTRAN:

```
count ENFORMRECEIVE ( ctlblock, buffer )
```

TAL:

```
[ count := ] ENFORMRECEIVE ( ctlblock , buffer )
```

where

`count`

returns a byte count for the length of the record retrieved (all FIND output records are the same length). A zero value means that all records have been returned.

`ctlblock`

INT:ref, is the same 18-word integer array control block that was supplied to ENFORMSTART for global storage among all ENFORM procedure calls. The host application program must not change the control block between calls to ENFORM.

`buffer`

INT:ref, is a pointer to the receiving buffer in the host application program for an output record. It must be at least as long as the value given for buffer-length in ENFORMSTART. The parameter *buffer* should be declared as FORTRAN record type if you want to access character information in a FORTRAN program.

Condition code settings:

> (CCG)	indicates that no more target records exist.
= (CCE)	indicates successful receipt of a target record.
< (CCL)	indicates a query processor error occurred; *buffer* contains additional error information, and *error-number* (passed to ENFORMSTART) contains an error number greater than zero.

FORTRAN programs must contain special code in order to examine the condition code settings.

ENFORMRECEIVE sends the records to the host language program, one at a time. Hence, ENFORMRECEIVE is repeatedly called until an end-of-file condition occurs, ENFORMFINISH is called, or an error condition occurs.

ENFORMRECEIVE Error Messages. If an error occurs during the execution of the ENFORM-RECEIVE Procedure, the number of the error is returned in error-number. Any of these error conditions terminate the ENFORM program. If the query processor is dedicated, it is deleted. Table 6-2 lists the error message for each of the possible error numbers. Error numbers with asterisks can have additional information returned for them in the buffer and are included in Table 6-3.

Table 6-2. ENFORMRECEIVE Error Messages

Number	Messages
1	Query processor received a message out of sequence.
3 *	Error occurred during file system READ.
4 *	Error occurred during file system WRITE.
5 *	Error occurred during file system POSITION or KEYPOSITION.
6	Error occurred during file system CONTROL.
7 *	Error occurred during SORT.
8 *	Error occurred during file system OPEN.
9	Error occurred during file CREATE.
10 *	Strategy exceeded the value of the @COST-TOLERANCE option variable.
11	An attempt was made to divide by zero.
12 *	Query contains an illegal combination of links.
14	Read limit exceeded value of the @READS option variable.
23	Ctlblock was modified by the host application program since last call to ENFORM.
25 *	There was a failure related to the use of an ENFORM server (process file).

Additional Information for ENFORM Error Messages. Some of the ENFORMRECEIVE errors (those indicated by an asterisk in Table 6-2) have additional information written to the receiving buffer specified by the *buffer* parameter. The buffer-length parameter for ENFORMSTART specifies the number of bytes of error information written to *buffer*. The additional error message has a maximum length of 30 bytes. If buffer-length is less than 30 bytes, the error information is truncated to whatever length was specified in buffer-length. The additional error messages are described in Table 6-3.

Table 6-3. Additional ENFORMRECEIVE Error Messages

Numbers	Additional Message
3-6,8	File System number: high-order byte of second word in binary format File Name (internal format): next 12 words contain the name of the physical file, in internal format, associated with the error
7	SORT error number: high-order byte of second word in binary format File Name (internal format): next 12 words contain the name of the physical file, in internal format, associated with the error File Error Code: next word in binary format
10	Actual ENFORM Strategy Cost for the query (number 1-8): high-order byte of second word in binary format
12	Error type: high order byte of second word in binary format. 0 At least one record has no LINK or WHERE clause relating it to any other record. Query will not be processed because a cross-product will result. See ERROR [92] in Appendix B for more information. 1 The record on the right side of a link optional is linked back to the record on the left side. See ERROR [178] in Appendix B for more information. 2 A record appears on the right side of more than one link optional. See ERROR [179] in Apopendix B for more information. Refer to the description of the LINK OPTIONAL statement in the *ENFORM Reference Manual* for a complete explanation of error types 2 and 3.
25	Error type: high-order byte of second word in binary format. 0 Error returned from server. Server error name and server error number contain a file name and an error number supplied by the server. 1-5 Error returned from Query Processor Server error name contains the file name of the server that caused the error. Server error number is not set. 1 Illegal dictionary description (see ERROR [112] in Appendix B) 2 Illegal use of KEY item (see ERROR [58] in Appendix B) 3 Illegal LINK field (see ERROR [68] in Appendix B) 4 Insufficient memory for buffer (see ERROR [50] in Appendix B) 5 Incorrect reply length (see ERROR [114] in Appendix B) Server error name: next 12 words contain a file name, in internal format, associated with the server. Server error number: next word in binary format.

ENFORMFINISH Procedure

ENFORMFINISH is called once to terminate the interface to ENFORM. The syntax of the ENFORMFINISH procedure is:

COBOL:

```
ENTER ENFORMFINISH USING ( ctlblock )
```

FORTRAN:

```
CALL ENFORMFINISH ( ctlblock )
```

TAL:

```
CALL ENFORMFINISH ( ctlblock )
```

where

`ctlblock`

INT:ref, is the same 18-word integer array control block that was supplied to ENFORM-START for global storage among all ENFORM procedure calls. The host language program must not change the control block between calls to ENFORM.

Examples

In the first example, a list of all the employees in a given region is required. A query containing one FIND statement is compiled. The query passes the required records to a COBOL program.

Before the COBOL program can execute, a description of the records being passed must be added to the data dictionary and the file containing the FIND statement must be compiled.

The record description of the record being passed is defined in the data dictionary as shown in Figure 6-2. The record description contains a field for the region number (*regnum*), the branch number (*branchnum*), and the employee name (*empname*). For complete instructions about dictionary definition, refer to the *Data Definition Language (DDL) Reference Manual.*

```
Record findfil.
  file is "findfil".
  02 regnum          type *.
  02 branchnum       type *.
  02 empname         pic x(18).
end
```

Figure 6-2. DDL Description of Record Passed to COBOL Program

The ENFORM query shown in Figure 6-3 contains only one FIND statement. The host language program builds a PARAM message containing the region number. Refer to the *Guardian Operating System Command Language and Utilities Manual* for the Command Interpreter PARAM message format.

```
OPEN employee;
OPEN branch;
OPEN findfil;
PARAM region-num;
LINK branch.primkey TO employee.dept;
FIND findfil
     (BY branch.regnum,
      BY branch.branchnum,
         employee.empname),
     WHERE branch.regnum EQUAL region-num;
```

Figure 6-3. Query Used to Pass Records to COBOL Program

The source query is in the Edit file, qfind. The following ?COMPILE command is used to compile the query and place it on the physical file, findfile.

```
?COMPILE qfind TO findfile
```

The COBOL host language program is shown in Figure 6-4. This program displays the target record as it is received from the query processor, unformatted.

```
IDENTIFICATION DIVISION.
  PROGRAM-ID. ENFORM-TEST.
  AUTHOR. E. TESTER.
  INSTALLATION.
  DATE-WRITTEN.      JUNE 1982.
  DATE-COMPILED.     JUNE 1982.

ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
  SOURCE-COMPUTER.   TANDEM/16.
  OBJECT-COMPUTER.   TANDEM/16.

DATA DIVISION.
  WORKING-STORAGE SECTION.
  01 END-FLAG                          PIC 9 VALUE 0.
     88 CLOSE-FROM-ENFORM               VALUE 1.
  01 NO-ENTRY                          PIC 9 VALUE 0.
     88 MORE-RECORDS-NEEDED             VALUE 0.
     88 NO-MORE-RECORDS-NEEDED          VALUE 1.
  01 TEMP-BINARY                       PIC S9999 COMP.
  01 TEMP-BINARY-BYTES                 REDEFINES TEMP-BINARY.
     02 TEMP-N1                        PIC X.
     02 TEMP-N2                        PIC X.
*ONE OF THE MOST USEFUL FEATURES OF A COBOL PROGRAM STARTING A
*COMPILED ENFORM QUERY IS THE ABILITY TO PASS PARAM VALUES.
*SEE ALSO THE A-INIT PARAGRAPH BELOW.
  01 PARAM-LIST.
     02 MESSAGE-TYPE                   PIC S9(4) COMP VALUE -3.
     02 NUM-PARAMS                     PIC S9(4) COMP VALUE 1.
     02 LENGTH-NAME                    PIC X.
     02 FILLER                         PIC X(10) VALUE "REGION-NUM".
     02 LENGTH-PARAM                   PIC X.
     02 REGION-NO                      PIC 9(4).
     02 LOW-NUMBER                     PIC S9(4) COMP VALUE 0.
  01 ENFORM-START.
     02 CTLBLOCK.
        04 CTLBLOCK-WORD               PIC XX OCCURS 18 TIMES.
     02 PHYSICAL-FILENAME              PIC X(24)
                                       VALUE "$MKT    SAMPLE   FINDFILE".
     02 BUFFER-LENGTH                  PIC 99 VALUE 80.
     02 ERROR-NUMBER                   PIC S9(4) COMP VALUE 0.
     02 RESTART-FLAG                   PIC S9(4) COMP VALUE 0.
  01 ENFORM-RECEIVE.
     02 DATA-COUNT                     PIC 9999 COMP VALUE 0.
     02 RECEIVED-DATA                  PIC X(80).

PROCEDURE DIVISION.
  MAIN SECTION.
BEGIN-COBOL-SERVER.
```

Figure 6-4. COBOL Host Language Program

```
    PERFORM A-INIT.
*THIS PROGRAM CALLS ENFORMSTART MULTIPLE TIMES UNTIL "9999"
*IS ENTERED.
    PERFORM B-TRANS UNTIL NO-MORE-RECORDS-NEEDED.
    PERFORM C-EOJ.
    STOP RUN.

 A-INIT.
*A PARAM MESSAGE TO A PROCESS REQUIRES TWO LENGTH BYTES; ONE FOR
*THE LENGTH OF THE PARAM NAME, AND ONE FOR THE LENGTH OF THE PARAM
*VALUE.
    MOVE 10 TO TEMP-BINARY.
    MOVE TEMP-N2 TO LENGTH-NAME.
    MOVE 4 TO TEMP-BINARY.
    MOVE TEMP-N2 TO LENGTH-PARAM.

B-TRANS.
    PERFORM B-START-ENFORM.
    IF MORE-RECORDS-NEEDED
       MOVE 1 TO RESTART-FLAG
       MOVE 0 TO END-FLAG
       PERFORM B-GET-RECORDS UNTIL CLOSE-FROM-ENFORM.

B-START-ENFORM.
    DISPLAY "PARAM VALUE = ".
    ACCEPT REGION-NO.
    IF REGION-NO EQUAL 9999
       MOVE 1 TO NO-ENTRY
    ELSE
       ENTER "ENFORMSTART" USING CTLBLOCK,
                                 PHYSICAL-FILENAME,
                                 BUFFER-LENGTH,
                                 ERROR-NUMBER,
                                 RESTART-FLAG,
                                 PARAM-LIST
       IF ERROR-NUMBER NOT EQUAL ZERO
          DISPLAY "ENFORMSTART ERROR: " ERROR-NUMBER
          MOVE 1 TO NO-ENTRY.
B-GET-RECORDS.
    ENTER "ENFORMRECEIVE" USING CTLBLOCK, RECEIVED-DATA
                          GIVING DATA-COUNT
*NOTICE THAT ERROR-NUMBER CAN BE CHECKED AFTER ENFORMRECEIVE EVEN
*THOUGH ERROR-NUMBER ITSELF IS NOT A PARAMETER TO ENFORMRECEIVE.
*THIS IS BECAUSE ENFORMSTART STORES THE ADDRESS OF ERROR-NUMBER
*IN CTLBLOCK.

    IF ERROR-NUMBER NOT EQUAL ZERO
       DISPLAY "ENFORMRECEIVE ERROR: " ERROR-NUMBER
       MOVE 1 TO END-FLAG
```

Figure 6-4. COBOL Host Language Program (Continued)

```
        ELSE
          IF DATA-COUNT EQUAL ZERO
           MOVE 1 TO END-FLAG
        ELSE
          DISPLAY RECEIVED-DATA.

      C-EOJ.
        DISPLAY "END OF RUN".
        MOVE 0 TO ERROR-NUMBER.
        ENTER "ENFORMFINISH" USING CTLBLOCK.
          IF ERROR-NUMBER NOT EQUAL ZERO
            DISPLAY "ENFORMFINISH ERROR: " ERROR-NUMBER.
```

Figure 6-4. COBOL Host Language Program (Continued)

In the second example, there is a need to do something with customers' orders for parts. Given a customer name and an order date, the part number and order number are returned for each part ordered by the customer on that date.

A TAL host language program is written to process those records. The ENFORM interface locates the desired records with considerably fewer lines of code than required by the host language.

Data is used from the *customer*, *order*, and *odetail* files of the sample relational data base. Refer to Appendix C.

The record description defined in the dictionary for the information returned to the host language program contains a field for the order number (*ordernum*) and part number (*partnum*). The DDL RECORD statement is shown in Figure 6-5.

```
RECORD order-process.
      05 ordernum        PIC 999.
      05 partnum         PIC 9(4).
END
```

Figure 6-5. DDL Description of Records Passed to TAL Program

The ENFORM query in Figure 6-6 contains one FIND statement and uses parameters specified by the user in Command Interpreter PARAM command (the PARAM command is passed to the TAL program in the Command Interpreter PARAM message). to communicate to the query processor the specific customer name and order date for the records to be processed. Refer to the *GUARDIAN Operating System Command Language and Utilities Manual* for the Command Interpreter PARAM message format. For each part number and order number returned via ENFORM-RECEIVE, some operation is performed. This portion of the example is left to your discretion.

```
OPEN customer,                              !Given a customer name and an
     order,                                 !order date, return one record
     odetail,                               !for each part ordered
     order-process;                         !containing the order number
LINK  customer TO order VIA custnum; !and the part number.
LINK  odetail TO order VIA ordernum;
PARAM passed-custname INTERNAL A18;
PARAM passed-orderdate INTERNAL A6;
FIND order-process
       (odetail.ordernum,
        odetail.partnum)
    WHERE custname    = passed-custname AND
          orderdate   = passed-orderdate;
```

Figure 6-6. An ENFORM Query for Host Language Interface

The source query, which resides in the Edit file, tfind, is compiled and placed in the physical file, *qryobj*, when the following is entered:

```
?COMPILE tfind TO qryobj
```

The TAL host application program that interfaces with ENFORM and processes the desired records is shown in Figure 6-7.

```
LITERAL  true = -1, false = 0;
 !filenames
INT      .query^filename[0:11] := "$MKT     SAMPLE   QRYOBJ   ";

 !global variables
INT      .cntl^block [0:17],        !ENFORM control block, 18 words
         .param^list [0:80];        !param list buffer
 !flag
INT      rec^processed;

 !variables
INT      error,
         count;
```

Figure 6-7. A TAL-Host Application Program Interfacing with ENFORM

```
! SCHEMA produced date - time : 4/21/82 12:15:59
! RECORD order-process created on 04/21/82 at 12:15
STRUCT      order^process^def (*);                              !DDL TAL output
  BEGIN
    STRUCT      odetail^key;
      BEGIN
        STRUCT      ordernum;
          BEGIN STRING BYTE [1:3]; END;
        STRUCT      partnum;
          BEGIN STRING BYTE [1:4]; END;
      END;
  END;

STRUCT  .order^process^rec (order^process^def);          !structure allocated here

?NOLIST
?SOURCE $system.system.extdecs (ENFORMSTART, ENFORMRECEIVE,
?           ENFORMFINISH, STOP);
?LIST

PROC get^params (param^message);
  INT .param^message;
BEGIN
      !This procedure obtains the Command Interpreter PARAM
      !message of the form provided by the Command
      !Interpreter. It will verify that exactly two parameters
      !have been supplied and those two parameters supply values for
      !passed-custname and passed-orderdate.
      !It also adds the REQUESTORS parameter required by the
      !query processor.
END;  !end of procedure get^params

  PROC main^proc MAIN;
  BEGIN
    CALL get^params (param^list);       !get parameters
                                        !start the ENFORM query
    CALL ENFORMSTART (cntl^block                !control block
                     ,query^filename            !compiled query
                     ,$len (order^process^rec), !buffer length
                     ,error                     !error number
                     ,                          !restart flag
                     ,param^list);              !parameter list
    DO BEGIN   !process all odetail records belonging to the order
       count := ENFORMRECEIVE (cntl^block           !control block
                              ,order^process^rec);  !buffer
      IF error <> 0 THEN                !report error to user
        CALL STOP;
      IF count THEN                     !if a record was received
        BEGIN
          rec^processed := true;        !received some information
          !process the information here
        END;                            !finished with this record
END UNTIL count = 0;                    !until no more odetail
                                        !records
    CALL ENFORMFINISH (cntl^block);     !control block
  END;              !end of main procedure
```

Figure 6-7. A TAL-Host Application Program Interfacing with ENFORM (Continued)

SECTION 7

ENFORM SERVERS

This section describes how to write and use ENFORM servers—processes that can supply data to the query processor as an alternative to the data being supplied directly from a disc file (Figure 7-1). ENFORM servers extend ENFORM's query capability by enabling you to use data that might otherwise be unusable by the query processor.

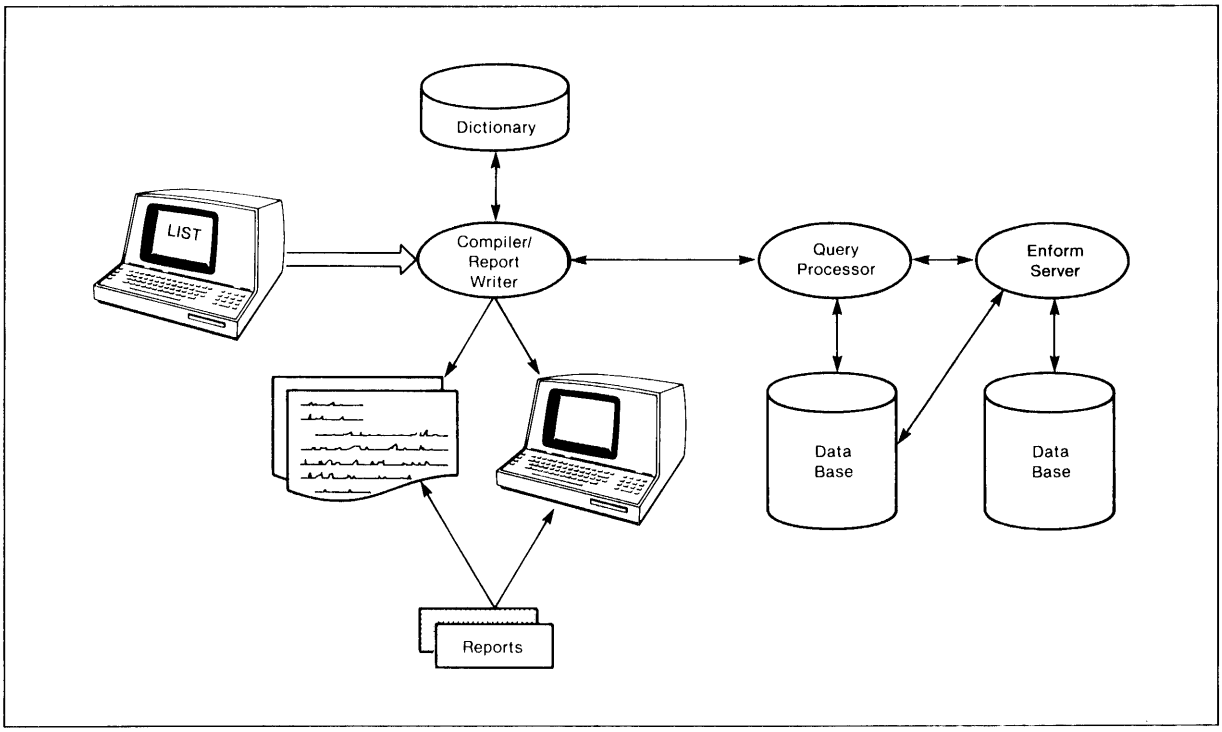


Figure 7-1. ENFORM Server Process.

An ENFORM server can reside on the same or on a different system from the query processor and can retrieve records from one or more data bases. However, query performance might be enhanced if the server resides on the same system as the data files the server reads.

An ENFORM server communicates through a series of interprocess messages with the query processor and appears to the query processor as a data file.

The query processor's access to an ENFORM server as a data file is limited to sequential access only; that is, one record at a time, starting with the first record, then the next and so on until the data is exhausted, or the query is satisfied. The query processor must be able to open the server more than once and must be able to to access the first and next records from the server repeatedly. There is no keyed or relative access defined for ENFORM servers.

Why Use ENFORM Servers

The following is a list of the types of data that create problems in query processing and the solution to the problems using ENFORM servers:

- Non-Relational Data

  Non-relational data refers to files containing data that is not normalized and, therefore, cannot be processed by ENFORM. An example of unnormalized data is an employee record containing a field representing an employee's dependents where the field occurs a variable number of times depending on another field in the record. For ENFORM to process the employee record, the dependents should be separated from the employee record and placed in a dependents file with either the employee's number or name specified as a key to identify the dependents.

  An ENFORM server can act as the dependents file by reading the records from a disc file, stripping the variable number of dependent records from each employee record, and then sending each dependent record to the query processor. This process can be repeated for each employee record until no more data is needed.

- Dirty Data

  Dirty data refers to data entered directly from a terminal. The data can contain blanks, control, and other characters that prevent ENFORM from making proper comparisons for data selection. An ENFORM server can act as a filter for a corresponding disc file by taking out the unacceptable characters and presenting clean records to the query processor.

- Concatenated Files

  For the purpose of reporting, you may want individual files with the same logical structure, which reside on one or more systems, to appear as one file to ENFORM. An ENFORM server can be written to access multiple files and return one record at a time to the Query Processor. The data supplied by the ENFORM server appears (to the query processor) to be coming from one file.

- Data Encryption

  An ENFORM server can be written to read a file containing encrypted data and to supply the data, in a decoded form, to the Query Processor.

- Data Compression

  An ENFORM server can be written to supply decompressed data from a compressed file and send the data to the query processor.

- Edit Files

  An ENFORM server can be written to read data from an edit file and send the data to the query processor.

- MUMPS

  MUMPS data stored in variable-length records cannot be directly processed by ENFORM. An ENFORM server can be written to read the variable-length records, restructure the data into fixed-field records, and then send the records to the query processor to be used with other data. Using an ENFORM server in this manner combines the convenience and compactness MUMPS provides for data storage with the querying and report writing facilities of ENFORM.

Some additional potential uses for an ENFORM server are: to convert data from one format to another and to provide some level of data independence by insulating applications from changes to the data base.

WRITING ENFORM SERVERS

ENFORM servers are process files. The rules for naming and identifying process files are described in the sections on file names and process files in the *GUARDIAN Operating System Programming Manual Volume 1 and Volume 2.*

The query processor identifies a specific ENFORM server through the server's process name. An ENFORM server is identified as a process by calling the process control procedure LOOKUP-PROCESSNAME with the file name. If the server is not a running named process, the call to LOOKUPROCESSNAME will be unsuccessful.

Servers are created by issuing the Command Interpreter RUN command or by a call to the NEWPROCESS procedure from a host language. The environment of a server is determined by the NEWPROCESS parameters and by information specified in the startup sequence. The NEWPROCESS parameters are used to assign the process name, to specify the execution priority and the cpu where the new process (an ENFORM server) executes. In addition, the startup sequence contains the default volume and subvolume names, and any ASSIGN and PARAM messages regarding the ENFORM server. (For more information on process control, refer to the *GUARDIAN Operating System Programming Manual Volume 1.)*

ENFORM Server and Query Processor Dialogue

Once an ENFORM server is identified and its environment is established, the server interacts with the query processor through a message dialogue to perform the following operations:

- Read the $RECEIVE file
- Interpret startup, open, initiate input, request input, terminate input, and close messages
- Identify and read records from one or more files
- Return reply messages and records to the query processor

The preceding dialogue takes place within an ENFORM server session. A session is one query execution or the period of time an ENFORM server is held open by a server query processor. (The ENFORM server can be designated as a file to be held open when the server query processor is started.)

An ENFORM server that is held open by a server query processor can be read many times for one request or many requests. An ENFORM server opened by either a dedicated or server query processor to process one query can also be read many times. In other words, once an ENFORM server is opened the data must be able to be read multiple times whether for one query or many queries.

Interprocess Communication

Communication between the query processor and an ENFORM server is based on a requester/server dialogue that enables a single round-trip message transfer. A message is transferred by the requester through a call to the GUARDIAN File Management procedure, WRITEREAD, against the file number of the server. A message is transferred by the server through a pair of calls to READUPDATE and REPLY (or their COBOL or FORTRAN equivalents) against $RECEIVE. One-way communcation—a READ instead of a READUPDATE call, will cause an error. Figure 7-2 illustrates two-way communication consisting of request and reply messages.

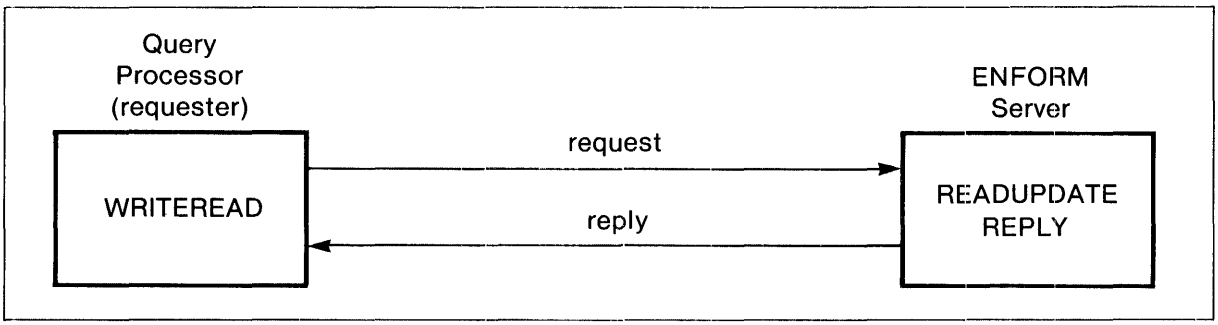


Figure 7-2. Query Processor and ENFORM Server Communication

The preceding discussion of interprocess communication assumes that you write ENFORM servers as "non-queuing servers", that is, servers that always reply to a request read from $RECEIVE before they read another request. Although they are potentially more complicated, you could, however, write "queuing servers"; that is, servers that sometimes will read a request from $RECEIVE before replying to the request previously read from $RECEIVE.

For more information on interprocess communication and queuing servers, refer to the *GUARDIAN Operating System Programming Manual.*

The interprocess messages used by the Command Interpreter (CI), the query processor (QP), and an ENFORM server are shown in Table 7-1. The effect of a message on an ENFORM server is reflected by a change in the condition of the server; that is, the message prepares the server for the next operation.

Table 7-1. ENFORM Server Session

Message Transfer	Server Condition
	Not Created
User Process CI — Call: NEWPROCESS; Startup: RUN ENFORM Server	Created Receives startup message. Dormant Waits for an OPEN.
Server Session Begins	
QP — OPEN: ENFORM Server	Opened Checks for message on $RECEIVE.
Server Application Code Begins	
QP — INITIATE-INPUT-REQUEST: ENFORM Server	Initializing QP identifies a server.
ENFORM Server — INITIATE-INPUT-REPLY: QP	Server reads request and replies.
QP — RECORD-INPUT-REQUEST: ENFORM Server	Active QP requests a record.
ENFORM Server — RECORD-INPUT-REPLY: QP	Server returns a record. These messages can be repeated many times.
QP — TERMINATE-INPUT-REQUEST: ENFORM Server	Terminating QP has no more requests.
ENFORM Server — TERMINATE-INPUT-REPLY: QP	Server returns to opened condition and can receive next INITIATE-INPUT-REQUEST.
Server Application Code Ends	
QP — CLOSE: ENFORM Server	Dormant Returns to dormant condition. Waits for an OPEN message.

Each of the messages in Table 7-1 is explained in the following section.

Message Protocol and Descriptions

The dialogue between processes is structured by a protocol that includes a startup message, an OPEN message, request and reply messages, and a CLOSE message. The messages transferred between processes to create, use, and terminate an ENFORM server are described below in the sequence of the protocol.

- Startup Message—The Command Interpreter startup message is sent to a process (an ENFORM server) when the process is successfully created. The startup message is used to define ENFORM server characteristics such as: the number of simultaneous requesters per server and the time period the ENFORM server waits before terminating, if there is no requester. One startup message is sent per ENFORM server creation. (For more information, refer to the *GUARDIAN Operating System Progamming Manual Volume 2*, Command Interpreter/Application Interface section.)

- OPEN Message—The OPEN message is a File Management Procedure used to establish a communication path between two processes; that is, the query processor and an ENFORM server. One OPEN message is sent per ENFORM server session. If you specify an exclusion mode via an ASSIGN command, that exclusion mode is passed in the OPEN message.

- Initialize Message—The initialize sequence is a pair (a request and a reply) of messages between the query processor and an ENFORM server. The initialize messages associate a query processor with an ENFORM server and describe the environment in which the ENFORM server performs. The query processor sends one initialize message sequence for each set of requests to an ENFORM server. More than one initialize message can be sent per ENFORM server OPEN message.

- Request Message—The query processor sends a request message to an ENFORM server to ask for a data record and the server replies with the record or an indication that there are no more records. The request message can contain data, end-of-file or error information. The query processor can send many request messages per ENFORM server session.

- Terminate Message—The terminate sequence is a pair (a request and a reply) of messages between the query processor and a server. The terminate message indicates there are no more requests for an ENFORM server. The query processor sends one terminate message per set of requests. More than one terminate message can be sent per ENFORM server OPEN message.

- CLOSE Message—The CLOSE message is a File Management Procedure used to terminate access to an ENFORM server. The CLOSE defines the end of use by the query processor and the end of an ENFORM server session. The query processor sends one CLOSE message per OPEN message.

The entire sequence of messages from OPEN through CLOSE can be performed for an ENFORM server more than once per query execution.

Message Components

The messages to and from the query processor and an ENFORM server consist of two components, a message header and the message data. The message header is a fixed length for all messages. The message data is a fixed length for all messages except RECORD-INPUT-REQUEST and RECORD-INPUT-REPLY; the lengths are determined during initialization.

DDL MESSAGE HEADER DESCRIPTION. The following DDL definition shows the message header format.

```
DEF pw-header-def.
    05  reply-code                      TYPE BINARY 16 .
!       = 1, indicates error, including end-of-file
!       = 0, everything's OK
    05 application-code                 PIC XX .
!       = "S1", indicating sequential file simulation
    05 function-code                    PIC XX .
!      = "DA" for data input
    05 trans-code                       PIC XX .
!      = "SR", initiate-input-request or -reply
!      = "RR", record-input-request or -reply
!      = "TR", terminate-input-request or -reply
    05 term-id                          PIC X(15) .
!      not used
    05 log-request                      PIC X .
!      not used
END


DEF ENFORM-error-header-def .
    05 error-code                       TYPE BINARY 16 .
!      = 0,  no error
!      = 1,  server EOF
!      = 29, invalid message or any other server detected error
    05 error-file-name                  PIC X(24) .
!      Must be blank unless error-code is non-zero.
!      If supplied, this filename will be printed in the error
!      message instead of the server's name.  Must be in the
!      form of a Tandem filename.
    05 file-error                       TYPE BINARY 16 .
!      Must be zero unless error-code is non-zero.
!      If supplied, this error number will be printed in the error
!      message instead of a file system error.
END


DEF ENFORM-server-header-def .
    05  pw-header-def                  TYPE * .
    05  ENFORM-error-header-def        TYPE *.
END
```

The message header is used to pass the following information between the query processor and an ENFORM server:

- Reply code — 0 indicates successful operation and 1 indicates an error.
- Application code — S1 indicates simple sequential file simulation.

NOTE

The application code, S1, identifies the message formats and protocol for the query processor and ENFORM server communication. This application code is passed in the header of every message to and from the query processor and a server. A new value for this field will be added when the protocol or message format changes to allow ENFORM to support multiple protocols.

- Function code — DA indicates data input.
- Transaction code — SR indicates an initiate input request or reply. RR indicates a record input request or reply. TR indicates a terminate input request or reply.
- Error code — Indicates one of the following internal error messages is returned from an ENFORM server to the query processor:

  0 no error

  1 end-of-file (only for RECORD-INPUT-REPLY message)

  29 missing parameter or invalid message format or any other server detected error (can be returned by any reply message)

- Filename — If an error code is other than zero and this field is not all blanks, this file name is printed in the error message instead of the server's name.
- File error — If supplied, an error number is printed in the error message instead of the GUARDIAN file error number. If filename is supplied, the file error must also be supplied. If filename is not supplied, this field is ignored.

NOTE

The DDL source file named ENFORMSV is supplied with the ENFORM product and can be used to generate message definitions for COBOL, FORTRAN, or TAL programs. The DDL source contains all of the message definitions necessary for a dialogue between the query processor and your ENFORM server.

ENFORM Server and Query Processor Messages

This section presents the message format and the DDL description for the following messages:

INITIATE-INPUT-REQUEST

INITIATE-INPUT-REPLY

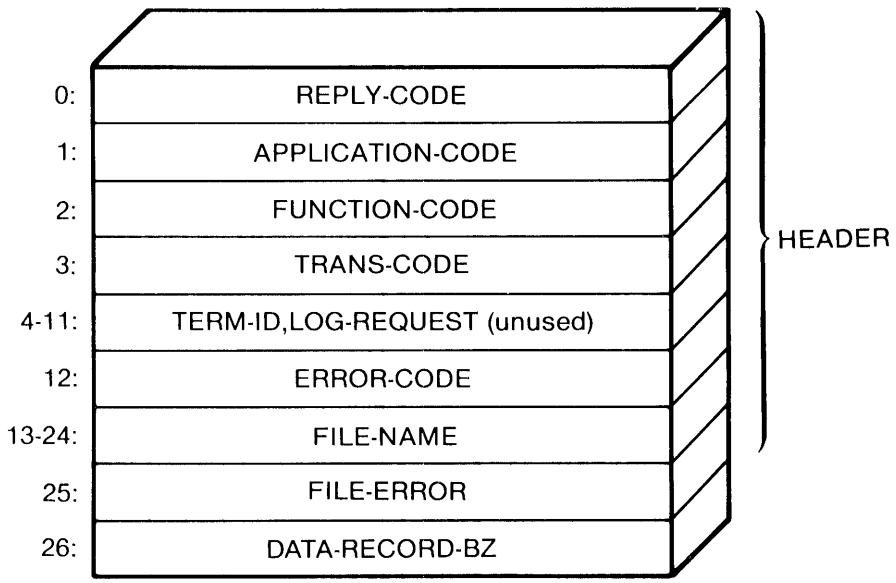
RECORD-INPUT-REQUEST

RECORD-INPUT-REPLY

TERMINATE-INPUT-REQUEST

TERMINATE-INPUT-REPLY

INITIATE-INPUT-REQUEST MESSAGE. The query processor sends this message to an ENFORM server before a request or set of requests for data. The message format and DDL description for the INITIATE-INPUT-REQUEST message are shown in Figure 7-3.




```
DEF  initiate-input-request-def .
     05  ENFORM-server-header-def        TYPE * .
     05  data-record-bz                  TYPE BINARY 16 .
!    data-record buffer size that will be sent on
!    record-input-request message and must remain
!    the same on record-input-reply messages. Size
!    is given in bytes and must be an even number.
END
```

Figure 7-3. Message Format and DDL Description for the INITIATE-INPUT-REQUEST Message

INITIATE-INPUT-REPLY MESSAGE. The ENFORM server returns this message to the query processor after reading an INITIATE-INPUT-REQUEST message. The message format and DDL description for the INITIATE-INPUT-REPLY message are shown in Figure 7-4.

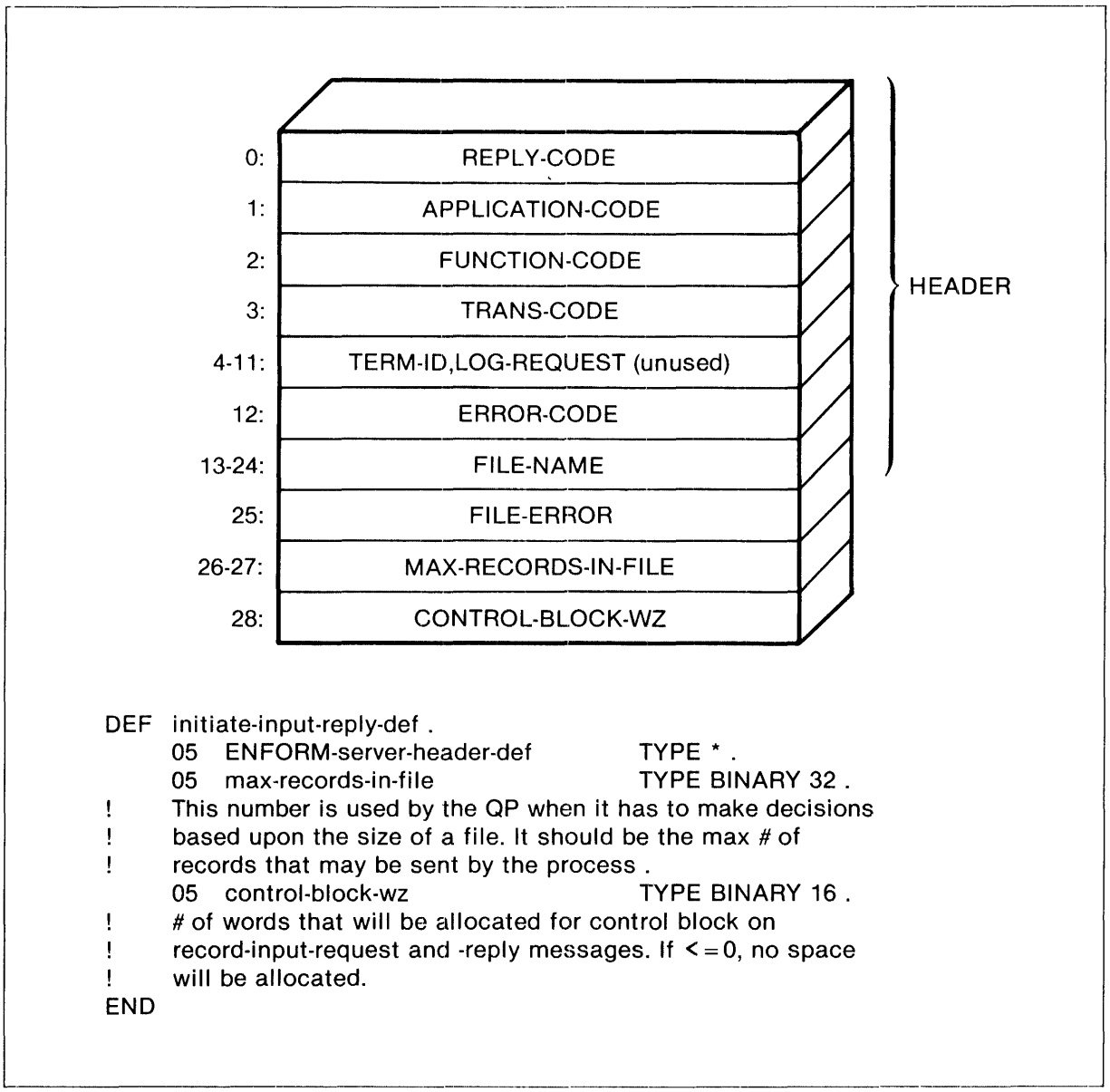
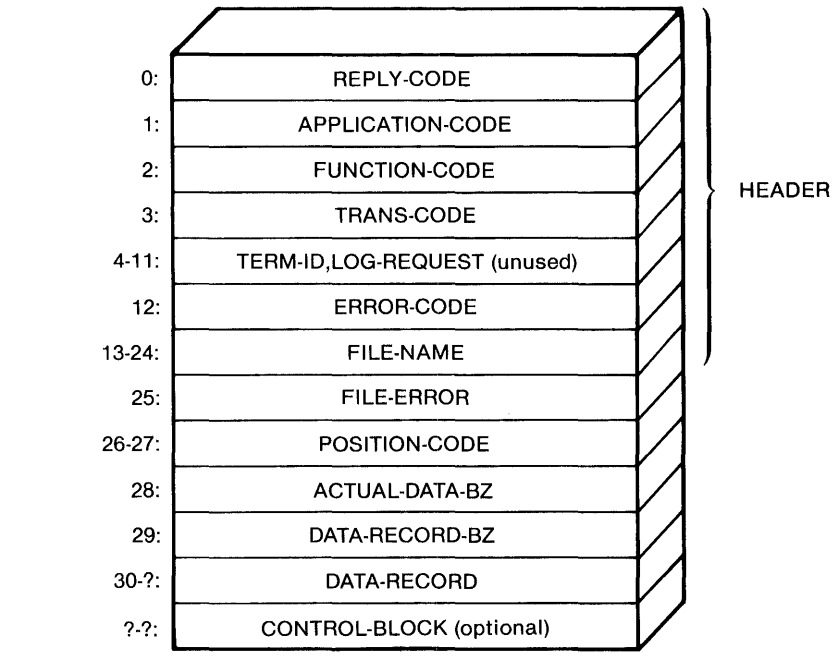


Figure 7-4. Message Format and DDL Description for the INITIATE-INPUT-REPLY Message

RECORD-INPUT-REQUEST MESSAGE. The query processor sends this message to the ENFORM server to ask for a data record. The message format and DDL description for the RECORD-INPUT-REQUEST message are shown in Figure 7-5.



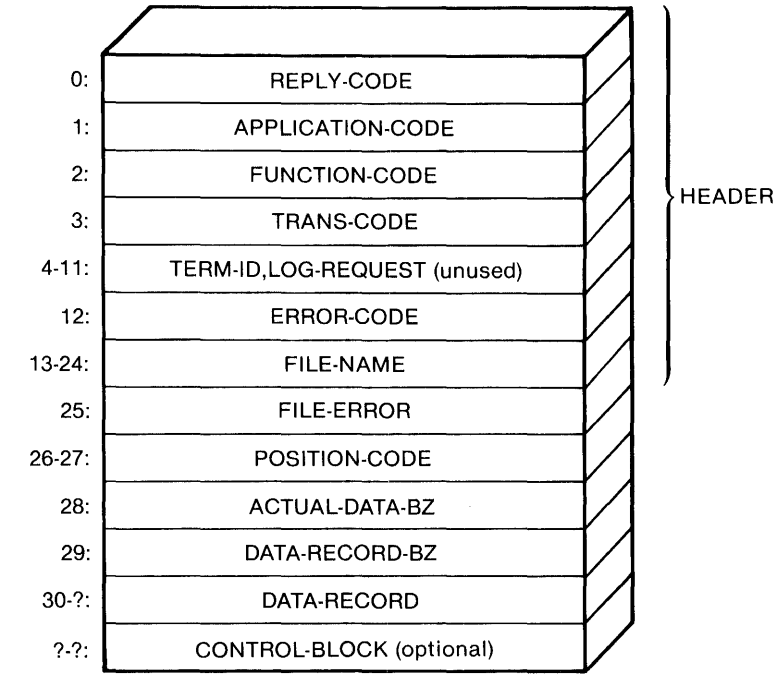

```
DEF  record-input-request-def .
     05  ENFORM-server-header-def          TYPE * .
     05  position-code                     PIC X(4) .
!        = "FRST", first record = "NEXT", next record
     05  actual-data-bz                    TYPE BINARY 16 .
!        = 1 to (MAX-DATA-WZ * 2) # bytes of actual data in the
!        DATA-RECORD field.
     05  data-record-bz                    TYPE BINARY 16 .
!        data-record field size which is fixed for the
!        duration of a session. Size is given in bytes and must
!        be an even number.
!    05  data-record                       PIC X(MAX-DATA-WZ * 2) .
     05  data-record                       PIC X(4096).
!        data-record field size is declared as the maximum data record
!        size. You can change it to be whatever your application
!        needs. This field contains whatever was sent over by a
!        previous record-input-reply or all zeroes if the first
!        request.
!    05  control-block                     PIC X(MAX-CB-WZ * 2) .
     05  control-block                     PIC X(4036).
!        ENFORM server context data or zeroes if first time
!        MAX-CB-WZ is the maximum control-block word size. You can
!        change it to be whatever your application needs.
END
```

Figure 7-5. Message Format and DDL Description for the RECORD-INPUT-REQUEST Message

The RECORD-INPUT-REQUEST message is bound by the following word size restrictions:

MAX-MSG-WZ	=	4096 words This is maximum size for RECORD-INPUT-REQUEST and RECORD-INPUT-REPLY messages.
MAX-DATA-WZ	=	2048 words This is the maximum key-sequenced ENSCRIBE record size.
MAX-MSG-HEADER-WZ	=	26 words This is ENFORM header word size.
MAX-CB-WZ	=	(4096-2048-26-4) = 2018 words This is the maximum user control block size.

RECORD-INPUT-REPLY MESSAGE. The ENFORM server returns this message to the query processor after reading a RECORD-INPUT-REQUEST message. The message format and DDL description for the RECORD-INPUT-REPLY message are shown in Figure 7-6.



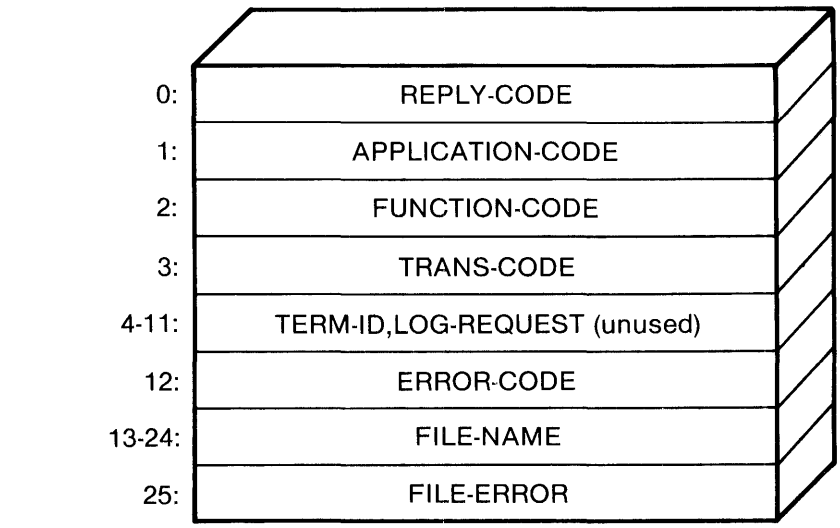

```
DEF  record-input-reply-def .
       05  ENFORM-server-header-def          TYPE * .
       05  position-code                     PIC X(4) .
!          = "FRST", first record = "NEXT", next record
       05  actual-data-bz                    TYPE BINARY 16 .
!          = 1 to (MAX-DATA-WZ * 2) # bytes of actual data in the
!          DATA-RECORD field.
       05  data-record-bz                    TYPE BINARY 16 .
!          data-record field size which is fixed for the
!          duration of a session. Size is given in bytes.
!      05  data-record                       PIC X(MAX-DATA-WZ * 2) .
       05  data-record                       PIC X(4096).
!          data-record field size is declared as the maximum data record
!          size. You can change it to be whatever your application
!          needs. This field contains the data record requested, either
!          the first or the next of the process file.
!      05  control-block                     PIC X(MAX-CB-WZ * 2) .
       05  control-block                     PIC X(4036).
!          ENFORM server context data. This field is optional.
!          MAX-CB-WZ is the maximum control-block word size. You can
!          change it to be whatever your application needs.
END
```

Figure 7-6. Message Format and DDL Description for the RECORD-INPUT-REPLY Message

The RECORD-INPUT-REPLY message is bound by the following word size restrictions:

MAX-MSG-WZ	=	4096 words This is maximum size for RECORD-INPUT-REQUEST and RECORD-INPUT-REPLY messages.
MAX-DATA-WZ	=	2048 words This is the maximum key-sequenced ENSCRIBE record size.
MAX-MSG-HEADER-WZ	=	26 words This is ENFORM header word size.
MAX-CB-WZ	=	(4096-2048-26-4) = 2018 words This is the maximum user control block size.

TERMINATE-INPUT-REQUEST MESSAGE. The query processor sends this message to an ENFORM server to indicate there are no more requests. The message format and DDL description for the TERMINATE-INPUT-REQUEST message are shown in Figure 7-7.




```
DEF  terminate-input-request-def .
     05   ENFORM-server-header-def          TYPE * .
END
```

Figure 7-7. Message Format and DDL Description for the TERMINATE-INPUT-REQUEST Message

TERMINATE-INPUT-REPLY MESSAGE. The ENFORM server returns this message to the query processor after reading a TERMINATE-INPUT-REQUEST message. The message format and DDL description for the TERMINATE-INPUT-REPLY message are shown in Figure 7-8.

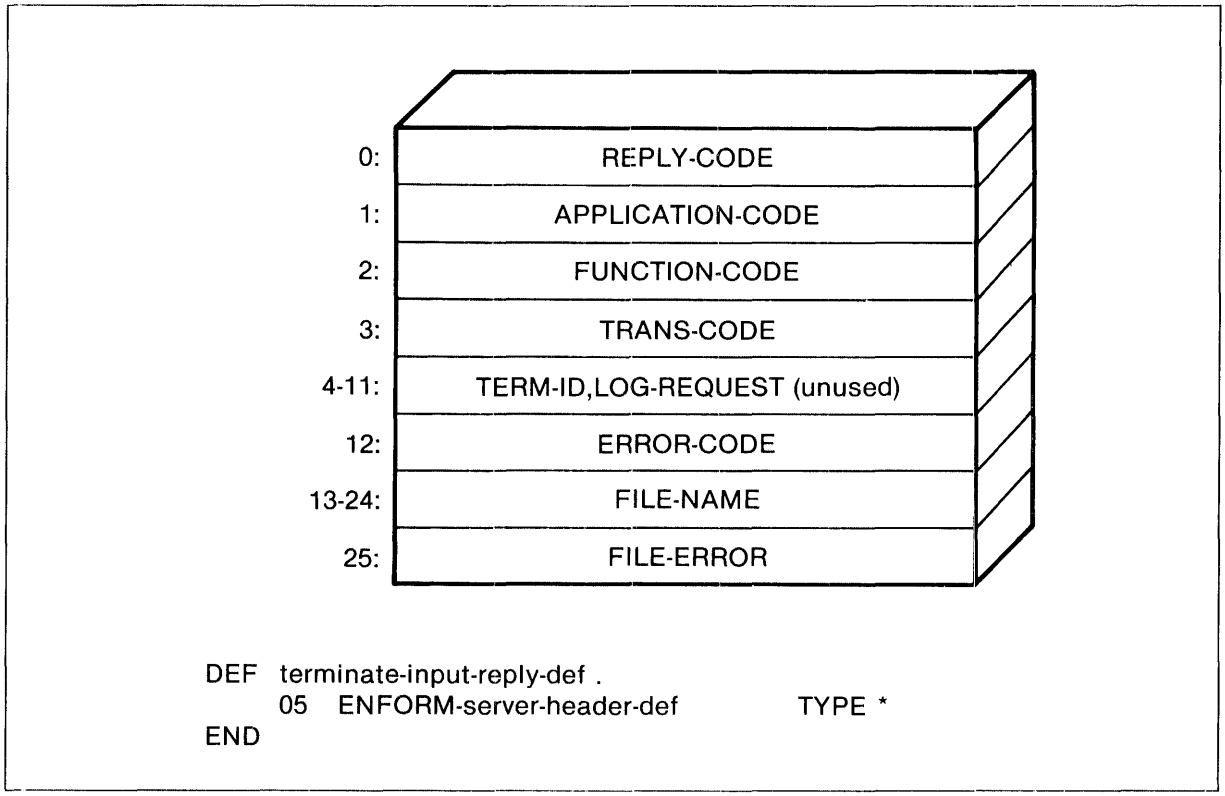


Figure 7-8. Message Format and DDL Description for the TERMINATE-INPUT-REPLY Message

ENFORM Server Operation—Restrictions and Conditions

When you write the server, consider the following:

- ENFORM servers can be used only for input to the query processor.

  Note that a process can be used for output data produced by ENFORM, but the protocol is different from and should not be confused with the ENFORM server protocol. For information on data produced by ENFORM, see the *ENFORM Reference Manual.*

- Access to an ENFORM server as data (a relation) by the query processor is limited to sequential access only.

  It might be necessary for the query processor to read a file more than once; for example, to process a LINK statement or when a NonStop query processor restarts a query. In this case, the ENFORM server must be able to send the query processor the first record, more than once, at any point during a series of request messages.

- An ENFORM server session consists of one query execution or the period of time an ENFORM server is held open by an active server query processor. An ENFORM server can be opened by multiple query processors at the same time. The ENFORM server can be designated as a file to be held open by the server query processor when the query processor is started.

  An ENFORM server can wait for a request without being opened by the query processor (requester).

- The message header is reinitialized and verified on every request and reply. The message data record area is not modified. The query processor initializes the control block to all zeros only on the first input request.

- If an ENFORM server returns an exception condition, or ENFORM encounters an exception condition while trying to communicate with an ENFORM server, the query processor attempts to CLOSE the ENFORM server.

- If a primary process failure occurs causing a server query processor to restart a query, the query begins by requesting the first record in the file. This operation is transparent to the ENFORM server, since an ENFORM server must be able to repeatedly supply records in a sequential manner.

- If an ASSIGN command is used to specify an exclusion mode for an ENFORM server, that exclusion mode is included in the OPEN message *flags* word *sysmsg[1].* Otherwise, the Query Processor provides the SHARED (0) exclusion mode. (For more information, refer to the *GUARDIAN Operating System Programming Manual Volume 1.*)

USING AN ENFORM SERVER

This section describes some general restrictions and performance considerations involved in using ENFORM servers.

Restrictions Related to Using ENFORM Servers

The following restrictions apply to the interaction of an ENFORM server with the query processor:

1. The ENFORM server must be running before the query processor issues an OPEN message. After receiving a Command Interpreter startup message, an ENFORM server can be opened by the query processor. It is your responsibility to start the ENFORM server before executing the ENFORM query that will use the server.

2. Document the order, in which the records are presented to the query processor, in the SEQUENCE IS clause of the related DDL description. This can prevent the query processor from having to read the ENFORM server file many times or perform an unnecessary sort operation.

3. A primary key or record key has no meaning for an ENFORM server. Using references such as KEY OF employee ... or employee.KEY is not allowed for records from an ENFORM server. Referring to a key in an ENFORM server file causes the query processor to issue an error message (Error 58).

4. The ENFORM server process name associated with a logical record name (a relation) can be specified in the FILE IS clause of the DDL record description, with a Command Interpreter ASSIGN command, or with an ENFORM ?ASSIGN command. The physical file name is used in the call to LOOKUPPROCESSNAME to determine whether or not the file is a process at the time a query is run.

5. The file type specified for an ENFORM server in the DDL description must be either UNSTRUCTURED or the file type must not be specified at all. If no file type is specified, the UNSTRUCTURED default is used. The query processor checks the file type at run time and returns an error message (Error 112) if a file type other than UNSTRUCTURED is indicated.

6. The query processor tries to allocate a message buffer for each ENFORM server relevant to a query. If memory space for the message buffer is not available, the query processor returns an error message (Error 50).

7. If an ENFORM server fails while being held open by a server query processor, the server query processor should be stopped and restarted. Since the query processor cannot determine from the system whether a process was stopped and restarted, merely restarting the ENFORM server does not ensure that the messages received by the server are in the proper sequence.

ENFORM Server Performance Considerations

The following are performance considerations related to using an ENFORM server.

QUALIFICATION. If there is a qualification (a WHERE clause) in a query and an ENFORM server represents the data for an item in the qualification, implement the qualification in the ENFORM server. In other words, the ENFORM server should send only the qualifying data to the query processor. A simple example is:

```
LIST employee.name WHERE employee.age > 65
```

where the ENFORM server supplies the employee records.

The ENFORM server can be written to supply the employee data such that only records containing employee names older than 65 are sent to the query processor. In this case, the query processor receives only records qualifying for the query instead of all the records.

ENFORM SERVER CONTEXT. ENFORM servers can be written as context-free or context-sensitive servers. The optional control block in the message format can be used to store context information. A data record received by the query processor in a request is returned in the next request and is available to the ENFORM server, as is the optional control block data. A control block contains all zeros the first time it is passed in a message to an ENFORM server.

VARIABLE-LENGTH DATA. The maximum size of the data is given to ENFORM by the dictionary. At initialization the query processor sends the maximum size (in bytes, rounded up to an even number) of the data buffer to the ENFORM server. This size remains fixed throughout the ENFORM server session.

The query processor is aware of records shorter than the length of the data buffer because the actual data size (in bytes) is returned from the ENFORM server with the data. A good practice is to have the ENFORM server blank or zero out unused space in the data buffer so the server does not confuse unused space with data.

If data records returned to the query processor are an odd byte length, the actual size of the data is smaller than the data buffer by at least one byte.

SERVER LOCATION. Query performance might be enhanced if the server resides on the same system as the data files the server reads.

ENFORM Server Example

The following example illustrates the general programming concepts associated with a simple ENFORM server program written in COBOL.

This ENFORM server accesses a file that contains information about parts and parts suppliers. The records are structured such that the part number is the primary key and each part can have up to 5 suppliers. This data is not in first normal form. The server returns records to the query processor one at a time in first normal form. Each record returned to the query processor contains the part number, one and ONLY one supplier number, and the cost of the part from that supplier.

Specific characteristics of the example ENFORM server include the following:

- The server is context free—the context information (the last part number and supplier number that was sent to the query processor) is returned in RECORD-DATA by the query processor. The optional control block area is used to save the occurrence number of the last supplier sent to the query processor.

- The server accepts up to 5 requesters—each of which should be checked for the sequence of its messages.

- The server does not check that each query processor sends messages in the correct sequence which is an initiate request, a record request, and then a terminate request.

```
IDENTIFICATION DIVISION.

PROGRAM-ID.  ENFSERV.
AUTHOR.   ANON.


ENVIRONMENT DIVISION.

CONFIGURATION SECTION.
SOURCE-COMPUTER.   TANDEM T/16.
OBJECT-COMPUTER.   TANDEM T/16.

INPUT-OUTPUT SECTION.
FILE-CONTROL.

    SELECT PARTSUP-FILE
           ASSIGN TO PRTS
           ORGANIZATION IS INDEXED
           ACCESS MODE IS DYNAMIC
           RECORD KEY IS PRIMKEY OF FROM-SUP
           FILE STATUS IS FILE-STAT.

    SELECT MESSAGE-IN
           ASSIGN TO $RECEIVE
           FILE STATUS IS RECEIVE-FILE-STATUS.

    SELECT MESSAGE-OUT
           ASSIGN TO $RECEIVE
           FILE STATUS IS RECEIVE-FILE-STATUS.

RECEIVE-CONTROL.
    TABLE OCCURS 5 TIMES
    SYNCDEPTH LIMIT IS 1
    REPLY CONTAINS MESSAGE-OUT RECORD.

DATA DIVISION.

FILE SECTION.
FD  PARTSUP-FILE
    LABEL RECORDS ARE OMITTED.
* Record FROM-SUP created on 03/05/82 at 09:43
 01 FROM-SUP.
   02 FRMSUP.
     03 PRIMKEY.
       04 PARTNUM                  PIC 9(4).
     03 NUM-SUPS                   PIC 9.
     03 SUP-DATA                   OCCURS 5 TIMES.
       04 SUPPNUM                  PIC 999.
       04 PARTCOST                 PIC 999999V99.

FD  MESSAGE-IN
    LABEL RECORDS ARE OMITTED
    RECORD CONTAINS 52 TO 78 CHARACTERS.
```

```
 01  MSG-INIT .
* Record INITIATE-INPUT-REQUEST created on 03/05/82 at 09:44
  05 INITIATE-INPUT-REQUEST.
    10 INITIATE-INPUT-REQUEST-DEF.
      15 ENFORM-SERVER-HEADER-DEF.
        20 PW-HEADER-DEF.
          25 REPLY-CODE              PIC S9(4)      COMP.
          25 APPLICATION-CODE        PIC XX.
          25 FUNCTION-CODE           PIC XX.
          25 TRANS-CODE              PIC XX.
          88 INIT-INPUT Value is "SR".
          88 RECORD-INPUT Value is "RR".
          88 TERMINATE-INPUT Value is "TR".
          25 TERM-ID                 PIC X(15).
          25 LOG-REQUEST             PIC X.
        20 ENFORM-ERROR-HEADER-DEF.
          25 ERROR-CODE              PIC S9(4)      COMP.
          25 ERROR-FILE-NAME         PIC X(24).
          25 FILE-ERROR              PIC S9(4)      COMP.
      15 DATA-RECORD-BZ              PIC S9(4)      COMP.

 01  MSG-RCD  .
* Record RECORD-INPUT-REQUEST created on 03/05/82 at 09:44
  05 RECORD-INPUT-REQUEST.
    10 RECORD-INPUT-REQUEST-DEF.
      15 ENFORM-SERVER-HEADER-DEF.
        20 PW-HEADER-DEF.
          25 REPLY-CODE              PIC S9(4)      COMP.
          25 APPLICATION-CODE        PIC XX.
          25 FUNCTION-CODE           PIC XX.
          25 TRANS-CODE              PIC XX.
          88 INIT-INPUT Value is "SR".
          88 RECORD-INPUT Value is "RR".
          88 TERMINATE-INPUT Value is "TR".
          25 TERM-ID                 PIC X(15).
          25 LOG-REQUEST             PIC X.
        20 ENFORM-ERROR-HEADER-DEF.
          25 ERROR-CODE              PIC S9(4)      COMP.
          25 ERROR-FILE-NAME         PIC X(24).
          25 FILE-ERROR              PIC S9(4)      COMP.
      15 POSITION-CODE               PIC X(4).
      88 FIRST-RCD Value is "FRST".
      88 NEXT-RCD Value is "NEXT".
      15 ACTUAL-DATA-BZ              PIC S9(4)      COMP.
      15 DATA-RECORD-BZ              PIC S9(4)      COMP.
      15 DATA-RECORD.
        20 FROMSUP.
          25 PRIMKEY.
            30 PARTNUM               PIC 9(4).
          25 SUPPNUM                 PIC 999.
          25 PARTCOST                PIC 999999V99.
          25 FILLER                  PIC X.
      15 CONTROL-BLOCK               PIC X(2).
      15 CB-INT REDEFINES CONTROL-BLOCK PIC 99.
```

```
 01  MSG-END  .
* Record TERMINATE-INPUT-REQUEST created on 03/05/82 at 09:44
  05 TERMINATE-INPUT-REQUEST.
    10 TERMINATE-INPUT-REQUEST-DEF.
      15 ENFORM-SERVER-HEADER-DEF.
        20 PW-HEADER-DEF.
          25 REPLY-CODE              PIC S9(4)      COMP.
          25 APPLICATION-CODE        PIC XX.
          25 FUNCTION-CODE           PIC XX.
          25 TRANS-CODE              PIC XX.
          88 INIT-INPUT Value is "SR".
          88 RECORD-INPUT Value is "RR".
          88 TERMINATE-INPUT Value is "TR".
          25 TERM-ID                 PIC X(15).
          25 LOG-REQUEST             PIC X.
        20 ENFORM-ERROR-HEADER-DEF.
          25 ERROR-CODE              PIC S9(4)      COMP.
          25 ERROR-FILE-NAME         PIC X(24).
          25 FILE-ERROR              PIC S9(4)      COMP.

 FD  MESSAGE-OUT
     LABEL RECORDS ARE OMITTED
     RECORD CONTAINS 52 TO 78 CHARACTERS.

 01  REPLY-INIT.
* Record INITIATE-INPUT-REPLY created on 03/05/82 at 09:44
  05 INITIATE-INPUT-REPLY.
    10 INITIATE-INPUT-REPLY-DEF.
      15 ENFORM-SERVER-HEADER-DEF.
        20 PW-HEADER-DEF.
          25 REPLY-CODE              PIC S9(4)      COMP.
          25 APPLICATION-CODE        PIC XX.
          25 FUNCTION-CODE           PIC XX.
          25 TRANS-CODE              PIC XX.
          88 INIT-INPUT Value is "SR".
          88 RECORD-INPUT Value is "RR".
          88 TERMINATE-INPUT Value is "TR".
          25 TERM-ID                 PIC X(15).
          25 LOG-REQUEST             PIC X.
        20 ENFORM-ERROR-HEADER-DEF.
          25 ERROR-CODE              PIC S9(4)      COMP.
          25 ERROR-FILE-NAME         PIC X(24).
          25 FILE-ERROR              PIC S9(4)      COMP.
      15 MAX-RECORDS-IN-FILE         PIC S9(9)      COMP.
      15 CONTROL-BLOCK-WZ            PIC S9(4)      COMP.
```

```
 01  REPLY-RCD
* Record RECORD-INPUT-REPLY created on 03/05/82 at 09:44
  05 RECORD-INPUT-REPLY.
    10 RECORD-INPUT-REPLY-DEF.
      15 ENFORM-SERVER-HEADER-DEF.
        20 PW-HEADER-DEF.
          25 REPLY-CODE              PIC S9(4)      COMP.
          25 APPLICATION-CODE        PIC XX.
          25 FUNCTION-CODE           PIC XX.
          25 TRANS-CODE              PIC XX.
          88 INIT-INPUT Value is "SR".
          88 RECORD-INPUT Value is "RR".
          88 TERMINATE-INPUT Value is "TR".
          25 TERM-ID                 PIC X(15).
          25 LOG-REQUEST             PIC X.
        20 ENFORM-ERROR-HEADER-DEF.
          25 ERROR-CODE              PIC S9(4)      COMP.
          25 ERROR-FILE-NAME         PIC X(24).
          25 FILE-ERROR              PIC S9(4)      COMP.
      15 POSITION-CODE               PIC X(4).
      88 FIRST-RCD Value is "FRST".
      88 NEXT-RCD Value is "NEXT".
      15 ACTUAL-DATA-BZ              PIC S9(4)      COMP.
      15 DATA-RECORD-BZ              PIC S9(4)      COMP.
      15 DATA-RECORD.
        20 FROMSUP.
          25 PRIMKEY.
            30 PARTNUM               PIC 9(4).
          25 SUPPNUM                 PIC 999.
          25 PARTCOST                PIC 999999V99.
          25 FILLER                  PIC X.
      15 CONTROL-BLOCK               PIC X(2).
      15 CB-INT REDEFINES CONTROL-BLOCK PIC 99.

 01  REPLY-END  .
* Record TERMINATE-INPUT-REPLY created on 03/05/82 at 09:44
  05 TERMINATE-INPUT-REPLY.
    10 TERMINATE-INPUT-REPLY-DEF.
      15 ENFORM-SERVER-HEADER-DEF.
        20 PW-HEADER-DEF.
          25 REPLY-CODE              PIC S9(4)      COMP.
          25 APPLICATION-CODE        PIC XX.
          25 FUNCTION-CODE           PIC XX.
          25 TRANS-CODE              PIC XX.
          88 INIT-INPUT Value is "SR".
          88 RECORD-INPUT Value is "RR".
          88 TERMINATE-INPUT Value is "TR".
          25 TERM-ID                 PIC X(15).
          25 LOG-REQUEST             PIC X.
        20 ENFORM-ERROR-HEADER-DEF.
          25 ERROR-CODE              PIC S9(4)      COMP.
          25 ERROR-FILE-NAME         PIC X(24).
          25 FILE-ERROR              PIC S9(4)      COMP.
```

```
WORKING-STORAGE SECTION.

77  I                          PIC S9(4) COMP VALUE 0.
77  FOUND-IT                   PIC S9(4) COMP VALUE 0.
77  ERROR-FN                   PIC X(24)
                                VALUE "$DATA   AASERV  PRTS   ".

01  REPLY-CODE-SET.
    05  OK-REPLY               PIC S9(4) COMP   VALUE 0.
    05  BAD-REPLY              PIC S9(4) COMP   VALUE 1.

01 ERROR-CODE-SET.
    05  ERR-NONE               PIC S9(4) COMP   VALUE 0.
    05  ERR-EOF                PIC S9(4) COMP   VALUE 1.
    05  ERR-INV                PIC S9(4) COMP VALUE 29.


  01  GUARD-FILE-ERRORS.
      05  FILLER               PIC S9(4)   COMP  VALUE 0.
      05  FILLER               PIC S9(4)   COMP  VALUE 1.
      05  FILLER                   PIC S9(4)   COMP  VALUE 59.
      05  FILLER                   PIC S9(4)   COMP  VALUE 46.
      05  FILLER                   PIC S9(4)   COMP  VALUE 10.
      05  FILLER                   PIC S9(4)   COMP  VALUE 11.
      05  FILLER                   PIC S9(4)   COMP  VALUE 45.
      05  FILLER                   PIC S9(4)   COMP  VALUE 45.

  01  GUARD-FILE-ERROR-TABLE REDEFINES GUARD-FILE-ERRORS.
      05  GUARD-FILE-ERR             PIC S9(4) COMP OCCURS 8 TIMES.

  01  FILE-DATA.
      05  ERR-INDEX                  PIC S9(4) COMP.
      05  FILE-ERROR-NO              PIC S9(4) COMP.
          88  END-OF-FILE            VALUE 1.
          88  FILE-IS-BAD            VALUE 59.
          88  INVALID-KEY            VALUE 46.
          88  DUPLICATE-KEY          VALUE 10.
          88  NO-EXISTING-RECORD     VALUE 11.
          88  FILE-IS-FULL           VALUE 45.
      05  RECEIVE-FILE-STATUS.
          10  STAT-1                 PIC 9.
              88  CLOSE-FROM-REQUESTOR  VALUE 1 THRU 3.
          10  STAT-2                 PIC 9.
      05  FILE-STAT.
          10  STAT-KEY1              PIC 9.
              88  NO-ERROR           VALUE 0.
              88  FILE-ERROR         VALUES ARE 1 THRU 3.
          10  STAT-KEY2              PIC 9.
      05  LOG-FILE-STATUS.
          10  S-KEY1                 PIC 9.
          10  S-KEY2                 PIC 9.
```

```
PROCEDURE DIVISION.

DECLARATIVES.

UA-PARTSUP-FILE SECTION.
   USE AFTER STANDARD ERROR PROCEDURE ON PARTSUP-FILE.
UA-PARTSUP-FILE-PROC.

*  FILE STATUS ALGORITHM.
*=================================================================
* STAT-KEY1    +  STAT-KEY2             = ERR-      ERROR CONDITION
*                                         INDEX
*      0       +       0        +   1   =   1    SUCCESSFUL COMPLETION
*      1       +       0        +   1   =   2    END OF FILE
*      3       +       0        +   0   =   3    PERMANENT ERROR
*      2       +       1        +   1   =   4    SEQUENCE ERROR
*      2       +       2        +   1   =   5    DUPLICATE KEY
*      2       +       3        +   1   =   6    NO EXISTING RECORD
*      2       +       4        +   1   =   7    PAST INDEXED EOF
*      3       +       4        +   1   =   8    PAST SEQUENTIAL EOF
*=================================================================

     IF STAT-KEY1 = 3 AND STAT-KEY2 = 0
        COMPUTE ERR-INDEX  =  STAT-KEY1 + STAT-KEY2
     ELSE
        COMPUTE ERR-INDEX  =  STAT-KEY1 + STAT-KEY2 + 1.

*=================================================================
*   If there is a file error, send the error number and the
*   filename in the reply buffer.  Also, set the reply code to
*   1 to indicate that an error occurred.  If the error was
*   an end of file (stat-key = 1) then set the error-code to 1;
*   otherwise set the error code to invalid.
*=================================================================

 MOVE GUARD-FILE-ERR (ERR-INDEX) TO FILE-ERROR OF REPLY-RCD.
 MOVE ERROR-FN TO ERROR-FILE-NAME OF REPLY-RCD.
 MOVE BAD-REPLY TO REPLY-CODE OF REPLY-RCD.
 IF STAT-KEY1 = 1
    MOVE ERR-EOF TO ERROR-CODE OF REPLY-RCD
 ELSE MOVE ERR-INV TO ERROR-CODE OF REPLY-RCD.

 UA-MESSAGE-IN SECTION.
     USE AFTER STANDARD ERROR PROCEDURE ON MESSAGE-IN.
 UA-MESSAGE-IN-PROC.
     IF STAT-1 IS NOT EQUAL 1
       DISPLAY "$RECEIVE FILE ERROR STATUS = ", RECEIVE-FILE-STATUS.

 UA-MESSAGE-OUT SECTION.
     USE AFTER STANDARD ERROR PROCEDURE ON MESSAGE-OUT.
 UA-MESSAGE-OUT-PROC.
     IF STAT-1 IS NOT EQUAL 1
       DISPLAY "$RECEIVE FILE ERROR STATUS = ", RECEIVE-FILE-STATUS.


 END DECLARATIVES.

 BEGIN-THE-PROGRAM.
     PERFORM START-YOUR-ENGINES.
     PERFORM READ-MAILBOX
         UNTIL CLOSE-FROM-REQUESTOR.
     PERFORM GARBAGE-COLLECTION.
     STOP RUN.
```

```
 START-YOUR-ENGINES.
     OPEN INPUT MESSAGE-IN.
     OPEN OUTPUT MESSAGE-OUT.
     OPEN INPUT PARTSUP-FILE EXCLUSIVE.

 GARBAGE-COLLECTION.
     CLOSE MESSAGE-IN.
     CLOSE MESSAGE-OUT.
     CLOSE PARTSUP-FILE.

 READ-MAILBOX.
*================================================================
*  Initialize the message and reply buffers, read $receive to
*  get the next message, and process the message.
*================================================================
     MOVE SPACES TO MSG-RCD, REPLY-RCD.
     MOVE ZERO TO FILE-STAT.
     PERFORM 90-GET-MESSAGE.
     IF NOT CLOSE-FROM-REQUESTOR
       PERFORM TRANS-CODE-CASE.

 TRANS-CODE-CASE.

     IF INIT-INPUT OF MSG-INIT
*          ****** initiate request ************
           PERFORM TRANS-CODE-SR

     ELSE IF RECORD-INPUT OF MSG-RCD
*            ********* record request *************
             PERFORM TRANS-CODE-RR

          ELSE IF TERMINATE-INPUT OF MSG-END
*              ******* terminate request *********
                  PERFORM TRANS-CODE-TR

               ELSE
*                       ******** error *************
                    MOVE ENFORM-SERVER-HEADER-DEF OF MSG-INIT TO
                          ENFORM-SERVER-HEADER-DEF OF REPLY-INIT
                    PERFORM 9-INVALID-MSG
                    PERFORM 90-SEND-END-REPLY.


 TRANS-CODE-SR.
*=========================================================
*    Initialize the header of the reply buffer.
*    The max number of records in the file = 100.
*    The size (in words) of the control block is 1.
*    Write the init reply.
*=========================================================

      MOVE ENFORM-SERVER-HEADER-DEF OF MSG-INIT TO
            ENFORM-SERVER-HEADER-DEF OF REPLY-INIT.
      MOVE 100 TO MAX-RECORDS-IN-FILE OF REPLY-INIT.
      MOVE 1 TO CONTROL-BLOCK-WZ.
      PERFORM 90-SEND-INIT-REPLY.
```

```
 TRANS-CODE-RR.
*=============================================================
*     Initialize the header of the reply buffer.
*     Initialize the position code, actual data size and data
*      record size and the data record in the reply buffer.
*     Process the request for the record and send the reply.
*=============================================================

      MOVE ENFORM-SERVER-HEADER-DEF OF MSG-RCD  TO
           ENFORM-SERVER-HEADER-DEF OF REPLY-RCD.
      MOVE POSITION-CODE OF MSG-RCD TO POSITION-CODE OF REPLY-RCD.
      MOVE ACTUAL-DATA-BZ OF MSG-RCD TO ACTUAL-DATA-BZ OF REPLY-RCD.
      MOVE DATA-RECORD-BZ OF MSG-RCD TO DATA-RECORD-BZ OF REPLY-RCD.
      MOVE SPACES TO DATA-RECORD OF REPLY-RCD.
      PERFORM 1-PROCESS-REQUEST.
      PERFORM 90-SEND-RCD-REPLY.

 TRANS-CODE-TR.
*=======================================================
*    Initialize the header of the reply buffer and
*    send the terminate reply.
*=======================================================

      MOVE ENFORM-SERVER-HEADER-DEF OF MSG-END  TO
           ENFORM-SERVER-HEADER-DEF OF REPLY-END.
      PERFORM 90-SEND-END-REPLY.

 1-PROCESS-REQUEST.
*====================================================================
*   Based on the value of the position code in the message buffer
*   get the first (or the next) record to return to the QP.
*====================================================================

      IF FIRST-RCD  OF MSG-RCD
        PERFORM 2-GET-FIRST-RECORD
      ELSE IF NEXT-RCD OF MSG-RCD
           PERFORM 2-GET-NEXT-RECORD
           ELSE PERFORM 9-INVALID-MSG.

 2-GET-FIRST-RECORD.
*=========================================================================
*     Read the first record in the file and return the part number
*     and the first occurrence of supplier and cost in that record.
*=========================================================================

      MOVE ZEROES TO PARTNUM OF FRMSUP.
      PERFORM 90-FIRST-POSITION.
      IF NO-ERROR
         PERFORM 90-READ-PARTSUP
         IF NO-ERROR
            MOVE 1 TO I
            PERFORM 9-FORMAT-REPLY.
```

```
 2-GET-NEXT-RECORD.
*=====================================================================
*  The QP returns the data that it was sent in the previous reply.
*  Use that partnum to position the file and read that record.  Then
*  compare the occurrence number of the last supplier sent (CB-INT)
*  to the number of suppliers in the record.  If = then read the next
*  record in the file and return the new part number and the first
*  occurrence of supplier in that record; otherwise, return the next
*  supplier in the same record. Any end of file indications are
*  handled by the declarative procedure.
*=====================================================================

      PERFORM 90-NEXT-POSITION.
      IF NO-ERROR
         PERFORM 90-READ-PARTSUP
         IF NO-ERROR
            IF CB-INT OF MSG-RCD LESS THAN NUM-SUPS
               COMPUTE I = CB-INT OF MSG-RCD + 1
               PERFORM 9-FORMAT-REPLY
            ELSE
               PERFORM 90-READ-PARTSUP
               IF NO-ERROR
                  MOVE 1 TO I
                  PERFORM 9-FORMAT-REPLY.


 9-FORMAT-REPLY.
*=====================================================================
*   Return the part number, the supplier number, and the part
*   cost.  Return the occurrence number of suppnum in the control
*   block. Also, set the number of bytes actually read to the
*   number of bytes in the record.
*=====================================================================

    MOVE PARTNUM OF FROM-SUP TO PARTNUM OF REPLY-RCD.
    MOVE SUPPNUM OF SUP-DATA(I) TO SUPPNUM OF REPLY-RCD.
    MOVE PARTCOST OF SUP-DATA(I) TO PARTCOST OF REPLY-RCD.
    MOVE I TO CB-INT OF REPLY-RCD.
    MOVE DATA-RECORD-BZ OF REPLY-RCD
                        TO ACTUAL-DATA-BZ OF REPLY-RCD.

 9-INVALID-MSG.

    MOVE BAD-REPLY TO REPLY-CODE OF REPLY-RCD.
    MOVE ERR-INV TO ERROR-CODE OF REPLY-RCD.

*=====================================================================
*
*          SERVER I/O ROUTINES
*
*=====================================================================

 90-GET-MESSAGE.
     READ MESSAGE-IN.

 90-FIRST-POSITION.
     START PARTSUP-FILE KEY NOT LESS THAN PARTNUM OF FRMSUP.
```

```
90-NEXT-POSITION.
    START PARTSUP-FILE KEY = PARTNUM OF FROM-SUP.

90-READ-PARTSUP.
    READ PARTSUP-FILE NEXT RECORD.

90-SEND-INIT-REPLY.
    WRITE REPLY-INIT.

90-SEND-RCD-REPLY.
    WRITE REPLY-RCD.

90-SEND-END-REPLY.
    WRITE REPLY-END.
```

APPENDIX A

SYNTAX SUMMARY

ENFORM syntax is summarized in this appendix. For specific details of syntax, refer to the language elements, statement, clause and command sections.

LANGUAGE ELEMENTS

Aggregates:

```
{ { AVG          }                                                          }
{ { COUNT        }  ( { field-name } [ OVER ALL       ]                    }
{ { MAX          }    { expression }  [ OVER over-item ]                    }
{ { MIN          }                                                          }
{ { SUM          }      [ WHERE logical expression ] ),                     }
{ { user-aggregate }                                                       }
{                                                                          }
{ { AVG          }                                                         }
{ { COUNT        }  ( [UNIQUE]   field-name  [ OVER ALL ]                  }
{ { MAX          }      [ WHERE logical expression ] ),                    }
{ { MIN          }                                                         }
{ { SUM          }                                                         }
{ { user-aggregate }                                                       }
```

Arithmetic operators:

```
+
-
*
/
```

IF/THEN/ELSE expression:

```
(IF logical expression THEN value-1 ELSE value-2)
```

Logical Expression:

```
[NOT] condition [ {AND} [NOT] condition ... ]
               [ {OR } ]
```

where condition has one of the following forms:

```
                  {BEGINS WITH                                  }
                  {  ']'                                        }
                  {CONTAINS                    string-literal   }
                  {  '>'                                        }
                  {conditional operator                         }
                  {                                             }
field-name        {  [NOT] {EQUAL}                              }
                  {        {EQ   }                              }
                  {        {IS   }          {value-range      } }
                  {        {=    }          {"["pattern-match"]"} }
                  {        {NE   }                              }
                  {        {<>   }                              }

{variable}                                  {variable}
{field-name} [NOT] conditional operator     {field-name}
{expression}                                {expression}
```

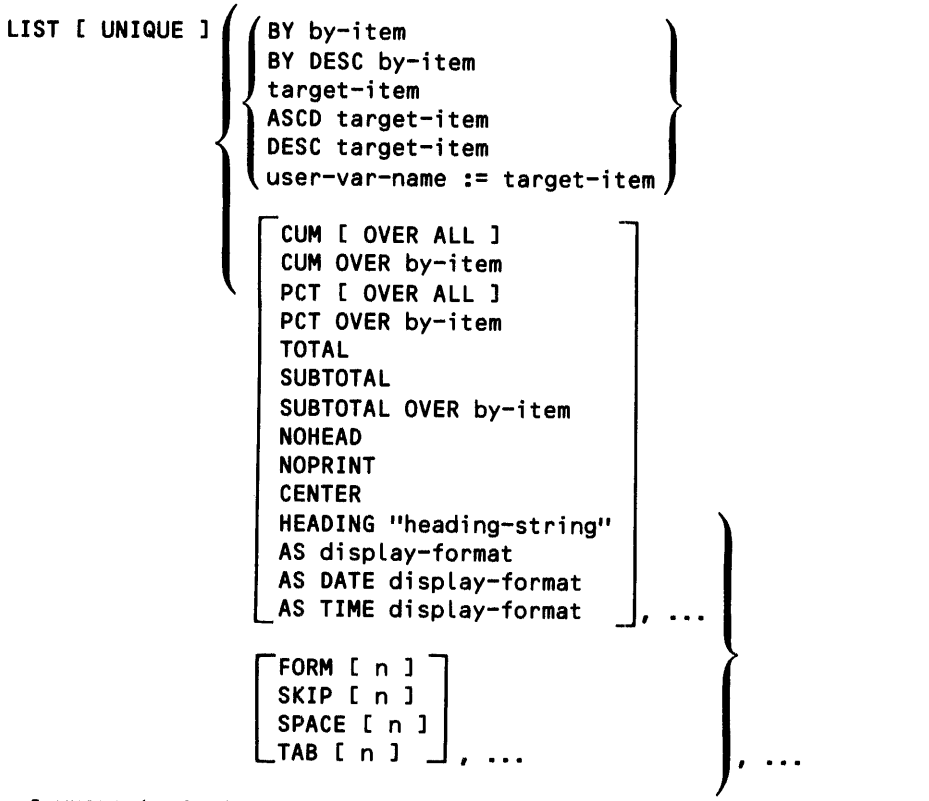
STATEMENTS

```
AT END [ PRINT print-list [ CENTER ] ] [ ; ]
```

```
AT START [ PRINT print-list [ CENTER ] ] [ ; ]
```

```
              {record-name        }
              {user-variable-name }
CLOSE         {user-aggregate-name} , ...[ ; ]
              {user-table-name    }
              {param-name         }
```

```
                  ( ( user-variable-name                                    )
                  ( ( user-table-name  "[" max-subscripts "]"               )
DECLARE           ( ( user-aggregate-name  ( formal-argument )              )
                  ( ( = ( step-expression [ , [ end-expression ]            )
                  (     [ , initialize-constant ] ] )                       )

                        [ INTERNAL internal-format ]
                        [ AS display-format ]                                  )
                        [ HEADING "heading-string" ]                           ),...[;]

DELINK ( record-name1 TO [ OPTIONAL ] record-name2 VIA field-name                )
       ( qualified-field-name1 TO [ OPTIONAL ] qualified-field-name2             ), ...[;]

DICTIONARY [ dict-subvol-name ] [ ; ]

EXIT [ ; ]

FIND [ UNIQUE ] output-record-name
       ( ( [ output-field-name := ] ( BY by-item                  )  )
       ( (                          ( BY DESC by-item             )  )
       ( (                          ( target-item                 )  )
       ( (                          ( ASCD target-item            )  )
       ( (                          ( DESC target-item            )  ), ... )
      [ WHERE logical-expression ] ;

FOOTING [ print-list [ CENTER ] ] [ ; ]

LINK ( record-name1 TO [ OPTIONAL ] record-name2 VIA field-name                  )
     ( qualified-field-name1 TO [ OPTIONAL ] qualified-field-name2               ), ...;
```




```
[ WHERE logical-expression ]

[ NOHEAD ALL ]
[ NOPRINT ALL ]
[ CENTER ALL ]

[ SUPPRESS [ WHERE ] logical expression ]
[ BEFORE CHANGE [ ON ] by-item PRINT print-list  [ CENTER ] ]
[ AFTER CHANGE [ ON ] by-item PRINT print-list   [ CENTER ] ]
[ AT START PRINT print-list                      [ CENTER ] ]
[ AT END PRINT print-list                        [ CENTER ] ]
[ TITLE print-list                               [ CENTER ] ]
[ SUBTITLE print-list                            [ CENTER ] ]
[ FOOTING print-list                             [ CENTER ] ]
[ SUBFOOTING print-list                          [ CENTER ] ]  ;
```

```
OPEN { record-name                                                    }
     { record-name2 [ AS ] COPY [ OF ] record-name1 } , ... [ ; ]

PARAM { param-name [ INTERNAL internal-format ] } , ... [ ; ]

SET { { user-variable-name                                  } TO { string-literal } }
    { { user-table-name"["subscript"]"                      }    { number         } } ,... [ ; ]
    { { param-name                                          }                       }
    {                                                               }
    {   option-variable-name  TO { ON             }                 }
    {                            { OFF            }                 }
    {                            { number         }                 }
    {                            { "character"    }                 }
    {                            { string-literal }                 }
    {                            { display-format }                 }

SUBFOOTING [ print-list [ CENTER ] ] [ ; ]

SUBTITLE [ print-list [ CENTER ] ] [ ; ]

TITLE [ print-list [ CENTER ] ] [ ; ]
```

CLAUSES

```
AFTER CHANGE [ ON ] by-item PRINT print-list [ CENTER ]

{ ASCD } field-name
{ DESC }

{ report-item  AS   [nonrepeatable-edit-descriptors] repeatable-edit-descriptors                }
{                                                                                               }
{ report-item  AS    " "[" [ decorations,... ] [ modifiers,... ] "]"                            }
{                       repeatable-edit-descriptors  "                                          }
{                                                                                               }
{ report-item  AS    " "[" [ decorations,... ] [ modifiers,... ] "]"                            }
{                         ( nonrepeatable-edit-descriptors                                      }
{                           repeatable-edit-descriptors )  "                                    }
```

where

report-item

is either a by-item or an target-item.

nonrepeatable-edit-descriptors

specify some general ways *report-items* are to be printed. *Nonrepeatble-edit-descriptors* should not be specified without a *repeatable-edit descriptor. Nonrepeatable-edit-descriptors* are:

P	multiplies value by 10**n, n is an integer.
S,SP,SS	for control of plus (+) sign printing.

repeatable-edit-descriptors

specify data conversion to the GUARDIAN Formatter for printing the *report-item* values. Valid values for *repeatable-edit-descriptors* are:

A [w]	for alphanumeric values.
Iw [.m]	for integer values.
Fw.d [.m]	for fixed point values.
M mask	for a template to combine literals and values.

where

w	specifies the width of the report-item.
m	specifies the number of digits that appear to the left of the decimal for fixed point values and the minimum number of digits for integer values.
d	specifies the number of digits to the right of the decimal.
mask	combination of the characters 9, Z, V, .(period) and literals. The combination must be enclosed within apostrophes (' ') or greater than and less than symbols (< >).

"["decorations"]"

specify character strings that can be added to a *report-item* depending on a condition. The syntax is:

conditions location char-string

where

conditions

are one or more of the following:

M	add *char-string* if value is negative.
N	add *char-string* if value is null.
P	add *char-string* if value is positive.
Z	add *char-string* if value is zero.
O	add *char-string* if overflow condition occurs.

location

is where the character string is to be printed:

An indicates *char-string* is to be printed at absolute position n.

F indicates *char-string* is to be inserted after the value is formatted. If *condition* is satisfied, *char-string* is printed immediately to the left of the item value.

P indicates *char-string* is inserted before the value is formatted. If *condition* is satisfied, *char-string* is prints to the right of the value.

char-string

is one or more alphanumeric characters enclosed within apostrophes (' ').

"["modifiers"]"

alter the effect of the edit descriptors as follows:

BN,BZ	prints blanks for null or zero values respectively
FL char	specifies a substitute fill character
OC char	respecifies the overflow character
LJ, RJ	specifies right or left justification
SS pr-of-symbols	allows substitution of symbols

where:

char

is an ASCII character enclosed in apostrophes.

pr-of-symbols

is a special mask symbol (see *repeatable edit-descriptors*) and a substitution character.

```
date-in-internal-format AS DATE { *              }
                                { display-format }

time-in-internal-format AS TIME { *              }
                                { display-format }

AT END PRINT print-list [ CENTER ]

AT START PRINT print-list [ CENTER ]

BEFORE CHANGE [ ON ] by-item PRINT print-list [ CENTER ]

{ BY      } field-name
{ BY DESC }
```

```
{ {target-item} CENTER }
{ {by-item    }        }
{                      }
{  CENTER ALL          }
```

```
target-item  CUM [OVER ALL    ]
                 [OVER by-item]
```

```
FOOTING  print-list [ CENTER ]
```

```
FORM [ number ]
```

```
{target-item} HEADING "heading-string"
{by-item    }
```

```
INTERNAL internal-format
```

```
JULIAN-DATE ( year, month, day )
```

```
{ {target-item} NOHEAD }
{ {by-item,   }        }
{                      }
{  NOHEAD ALL          }
```

```
{ {target-item} NOPRINT }
{ {by-item    }         }
{                       }
{  NOPRINT ALL          }
```

The Option Variables and their legal values are:

```
{ @BLANK-WHEN-ZERO }
{ @BREAK-KEY       }
{ @CENTER-PAGE     } TO {ON }
{ @HEADING         }    {OFF}
{ @STATS           }
{ @SUMMARY-ONLY    }
{ @WARN            }
```

```
{ @COPIES                }
{ @COST-TOLERANCE        }
{ @DISPLAY-COUNT         }
{ @LINES                 }
{ @MARGIN                }
{ @PAGES                 }
{ @PRIMARY-EXTENT-SIZE   } TO number
{ @SECONDARY-EXTENT-SIZE }
{ @READS                 }
{ @SPACE                 }
{ @TARGET-RECORDS        }
{ @VSPACE                }
{ @WIDTH                 }
```

```
{ @DECIMAL           }
{ @NEWLINE           } TO "character"
{ @NONPRINT-REPLACE  }
{ @OVERFLOW          }
{ @UNDERLINE         }

  @SUBTOTAL-LABEL  TO "char-string"

{ @DATE-FORMAT } TO display-format
{ @TIME-FORMAT }

target-item  PCT [OVER ALL    ]
                 [OVER by-item]

SKIP [ number ]

SPACE [ number ]

SUBFOOTING print-list [ CENTER ]

SUBTITLE print-list [ CENTER ]

target-item  SUBTOTAL [OVER by-item]
                      [OVER ALL    ]

SUPPRESS [ WHERE ] logical-expression

System Variables:

  @DATE
  @TIME
  @LINENO
  @PAGENO

TAB [ number ]

TIMESTAMP-DATE ( field-name )

TIMESTAMP-TIME ( field-name )
```

```
TITLE print-list [ CENTER ]

{target-item  TOTAL}
{by-item           }

WHERE logical-expression
```

COMMANDS

```
?ASSIGN [{record-name       } [,] [TO] physical-filename
        {generic-file-name}]
        {, create-open spec} ... ]

?ASSIGN  record-name , { , create-open spec } ...

?ATTACH [ process-name ]

?COMPILE edit-filename [ ( section-name, ... ) ] TO compiled-physical-filename

?DICTIONARY  [ dict-subvol-name ]

?EDIT [ edit-filename ]

?EXECUTE  compiled-physical-filename

?EXIT

?HELP [ help-element ]

?OUT [ physical-filename ]

?RUN [ edit-filename [ ( section-name, ... ) ]

?SECTION section-name

          [OPEN                      ]
          [LINK                      ]
          [CONTROL                   ]
?SHOW     [LIMITS                    ]
          [ASSIGN [ record-name ]    ]
          [user-variable-name        ]
          [record-name               ]
          [param-name                ]

?SOURCE edit-filename [ ( section-name, ... ) ]
```

ENFORM PROCEDURES

COBOL:

```
ENTER ENFORMSTART USING  ctlblock                      !INT:ref
                       , compiled-physical-filename    !INT:ref
                       , buffer-length                 !INT:value
                       , error-number                  !INT:ref
                       [ , restart-flag ]              !INT:value
                       [ , param-list ]                !INT:ref
                       [ , assign-list ]               !INT:ref
                       [ , process-name ]              !INT:ref
                       [ , cpu ]                       !INT:value
                       [ , priority ]                  !INT:value
                       [ , timeout ]                   !INT:32:value
                       [ , reserved-for-expansion ]    !INT:ref

ENTER ENFORMRECEIVE USING ctlblock, buffer [ GIVING count ]
                          !INT:ref  INT:ref         INT:function!

ENTER ENFORMFINISH USING ( ctlblock )                  !INT:ref
```

FORTRAN:

```
CALL ENFORMSTART (ctlblock                        !INT:ref
                 ,compiled-physical-filename      !INT:ref
                 ,\buffer-length\                 !INT:value
                 ,error-number                    !INT:ref
                 [, \restart-flag\]               !INT:value
                 [, param-list ]                  !INT:ref
                 [, assign-list ]                 !INT:ref
                 [, process-name ]                !INT:ref
                 [, \cpu\ ]                       !INT:value
                 [, \priority\ ]                  !INT:value
                 [, \timeout\ ]                   !INT:32:value
                 [, reserved-for-expansion ]      !INT:ref
                 ,\maskword\ )                    !INT:value

count ENFORMRECEIVE ( ctlblock, buffer )
!INT:function         INT:ref   INT:ref  !

CALL ENFORMFINISH ( ctlblock )                    !INT:ref
```

TAL:

```
CALL ENFORMSTART ( ctlblock                     !INT:ref
                 , compiled-physical-filename   !INT:ref
                 , buffer-length                !INT:value
                 , error-number                 !INT:ref
                 [ , restart-flag ]             !INT:value
                 [ , param-list ]               !INT:ref
                 [ , assign-list ]              !INT:ref
                 [ , process-name ]             !INT:ref
                 [ , cpu ]                      !INT:value
                 [ , priority ]                 !INT:value
                 [ , timeout ]                  !INT:32:value
                 [ , reserved-for-expansion ]) !INT:ref

[ count := ] ENFORMRECEIVE ( ctlblock , buffer )
 !INT:function               INT:ref    INT:ref !

ENFORMFINISH ( ctlblock )                        !INT:ref
```

APPENDIX B

ERROR MESSAGES

This appendix documents the following types of messages:

- *!!! ERROR error-number* types: mean a serious error has occurred. Statement execution terminates. If this type of error occurs for a LIST or FIND statement, the query terminates.
- **** WARNING warning-number* types: point out an error that could change the expected results. The error does not abort the query although it could lead to more serious error conditions.
- **** FILE ERROR ...* types: mean a serious error has occurred within the file system. If there is a file error with the run-time IN input file, the dictionary file, or the vocabulary file, then the entire ENFORM session is terminated.
- **** ...* types: occur during ENFORM initialization. If this type of error occurs, ENFORM terminates abnormally.
- *ENFORM [QP] TRAP* means a that either a hardware failure or an unexpected software error has occurred. Please save the information produced by this message and report the error to Tandem.
- **** ERROR* types: means an error has occurred during execution of the BUILDMK utility. BUILDMK terminates abnormally. Correct the problem and rerun BUILDMK or you cannot use the key-sequenced version of the message table with ENFORM.

Error message are listed in the following order within this appendix:

1. ENFORM initialization errors are listed in alphabetical order.
2. *!!! ERROR* and **** WARNING* type errors are listed together in numeric order with the error message text and additional comments.
3. **** FILE ERROR* type errors are described in alphabetical order.
4. *ENFORM TRAP* messages are listed in alphabetical order.
5. BUILDMK error messages are listed. These messages consist of the following types of messages: *!!!ERROR* and *FILE ERROR* messages.

ENFORM INITIALIZATION MESSAGES

*** Current reserved word cannot be used to redefine another reserved word

A reserved word redefinition in the ?VOCABULARY section of the message table contains an old reserved word where a new word is expected. (The key-sequenced message table file was not built by the BUILDMK utility or the file has been modified since it was built).

*** Invalid DICTIONARY file name

The dictionary file name specified on the ENFORM command line is not a valid GUARDIAN file name.

*** Invalid MESSAGE TABLE file name

The message table file name specified on the ENFORM command line is not a valid GUARDIAN file name.

*** Message table does not contain a version number record. Rebuild key-sequenced file

Either the key-sequenced message table file was not built by the BUILDMK utility or the file has been modified since it was built.

*** Message table must be a disk file

Self-explanatory. Use the BUILDMK utility to build the key-sequenced message table file.

*** Message table must be a key-sequenced file

Self-explanatory. Use the BUILDMK utility to build the key-sequenced message table file.

*** Message table must contain both ?MESSAGES and ?HELP sections

Self-explanatory. Use the BUILDMK utility to build the key-sequenced message table file.

*** Message table version number is not correct

The version number in the message table does not match the version number expected by ENFORM. Rebuild the key-sequenced message table file using the appropriate version of BUILDMK.

*** Primary key for message table file must be offset at 0 and have length 34

Self-explanatory. Use the BUILDMK utility to build the key-sequenced message table file.

*** Sorry, you're not allowed to run ENFORM on this processor

This processor does not have the required ENFORM microcode.

!!! ERROR AND *** WARNING TYPE MESSAGES

```
!!! ERROR [26] Invalid use of range item
```

A subscript range may only be used as a target-item (but cannot be used when modified by a BY, BY DESC, ASCD, or DESC clause) in a LIST or FIND statement. Its use is invalid in all other circumstances.

```
!!! ERROR [27] Unknown ENFORM directive or syntactically incorrect
```

A command name has been misspelled or attempt has been made to execute a command from a different subsystem.

```
!!! ERROR [28] The boolean operators AND and OR cannot be used in a TITLE or PRINT
    statement expression
```

Only a simple logical expression may be used in a IF/THEN/ELSE expression within an AFTER CHANGE, AT END, AT START, BEFORE CHANGE, FOOTING, SUBFOOTING, SUBTITLE, or TITLE statement or clause.

```
!!! ERROR [29] Reference has been attempted to an undefined or illegal item in PRINT
    statement
```

An item has been used in a Print List clause that does not appear in the LIST statement.

```
*** WARNING [30] Name too long. Truncated to 31 characters
```

The variable or aggregate name must be less than 31 characters.

```
!!! ERROR [31] Invalid file name
```

The file name printed with this error message is not a valid Tandem physical file name.

```
!!! ERROR [32] Invalid SET-variable specification
```

The '@' symbol is not followed by a legal string for an option variable name.

```
!!! ERROR [33] Name not found
```

A field name has been misspelled or the wrong dictionary is being used.

```
!!! ERROR [34] Name not sufficiently qualified to avoid ambiguity
```

Field name appears in more than one opened record description. Use more qualification.

```
!!! ERROR [35] Record description not found in dictionary
```

The record name has been misspelled or the wrong dictionary is being used.

```
!!! ERROR [36] Symbol table overflow
```

The maximum available space for file descriptions, user defined variables, etc., has been exceeded. All tables are cleared.

```
!!! ERROR [37] Overflow encountered on input conversion of numeric-literal
```

Numeric literal exceeds 32767.

```
!!! ERROR [38] ?SOURCE file nesting > 4
```

Must have four or fewer levels of nested ?SOURCE commands.

```
!!! ERROR [39] Too many references to user aggregates
```

Total number of references to user aggregates exceeds 32.

```
!!! ERROR [40] Multiply defined name
```

The name already exists as a record name, user-defined item, or parameter name.

```
!!! ERROR [41] The maximum target length of 2000 bytes was exceeded.
Unable to process query
```

The maximum length of a LIST or FIND statement requiring sorting exceeded 2000 bytes, or the maximum length of a LIST or FIND statement without sorting requirements exceeded 4095 bytes.

```
!!! ERROR [42] Expression too large to process
```

Expression must contain fewer than 512 items.

```
!!! ERROR [43] The specified relation is invalid in the above context
```

CONTAINS, BEGINS WITH, and pattern match conditions require string arguments. The pattern match operation allows only EQ and NE operators.

```
!!! ERROR [44] Too many actual file assignments
```

Table of assignments exceeds eight entries. Clear the table by entering ?ASSIGN without a physical file name.

```
!!! ERROR [45] An integer literal is required in the above context
```

A number with decimal places is not allowed. Must be an integer.

```
!!! ERROR [46] Too many LINKs
```

Number of links exceeds 32. Clear some with a DELINK statement.

```
*** WARNING [47] Source line was truncated
```

Line must be 255 characters or less.

```
!!! ERROR [48] Only field names may appear in a qualification
```

The item name in the WHERE or SUPPRESS clause has not been defined or is misspelled. This is usually due to an internal error.

```
!!! ERROR [49] User variable assignments are illegal in the scope of a FIND statement
```

User-defined variable or table is not an acceptable output field name in a FIND statement.

```
!!! ERROR [50] Insufficient memory available for data buffer (SERVER-related failure
on name)
```

An ENFORM server (process file) cannot be opened because there is no space for a message buffer.

*** WARNING [51] Null target list

LIST or FIND statement is not processed.

!!! ERROR [52] Not currently supported

The indicated feature or operation may not be used.

!!! ERROR [53] Invalid subscript range specification

The subscript range specified ([x:y]) for an item is wrong. Subscripts must be numeric literals. The first number (x) must be smaller than the second number (y).

!!! ERROR [54] Invalid AS format description

The display format for AS, AS DATE, or AS TIME is invalid or the INTERNAL format is invalid.

!!! ERROR [55] A user aggregate may not be used in a user aggregate end-expression

Self-explanatory.

!!! ERROR [56] An aggregate may not be used as the argument to another aggregate

Self-explanatory.

!!! ERROR [57] Item type incompatible with use

Expecting a field or user defined variable to subscript or illegal use of a condition in an Arithmetic Expression clause. Similar to a type mismatch.

!!! ERROR [58] Illegal use of KEY item (SERVER-related failure on name)

Record-name.KEY or KEY OF record-name is not allowed when the data for record-name is from an ENFORM server (process file).

!!! ERROR [59] Maximum read count exceeded

ENFORM has read the limit number of records specified by the @READS Option Variable clause.

!!! ERROR [60] A user aggregate declaration may not reference the value of another user aggregate

Self-explanatory.

!!! ERROR [61] Initialization expression must be numeric

Attempted to initialize a user-defined aggregate with something other than a number or using JULIAN-DATE clause within a SET statement that does not evaluate to a literal.

!!! ERROR [62] Too many target items

Number of items or output fields within LIST or FIND statement exceeds 400.

!!! ERROR [63] Too many PRINT statements

Number of items in Print List clauses exceeds 172. Processing of the ENFORM program is stopped and the contents of the internal table is reset to the values held at start of processing the statement which produced the error.

!!! ERROR [64] By-item not found

Either the grouped item was not defined in a BY or BY DESC clause or the item was misspelled.

!!! ERROR [65] An aggregate may not be used in a print-list clause.

Self-explanatory.

!!! ERROR [67] Field type incompatibility

Data types being compared must both be numeric or alphabetic.

!!! ERROR [68] Illegal LINK field

Misspelled qualified field name or attempted to use a subscripted field where not allowed.

!!! ERROR [68] Illegal LINK field (SERVER-related failure on name)

Attempted to use an ENFORM server (process file) improperly in a LINK OPTIONAL statement. For example, LINK A TO OPTIONAL B ..., A cannot be an ENFORM server because of an implementation restriction.

!!! ERROR [69] Invalid range

Incorrectly defined a range for a comparison pattern or THRU within a Logical Expression clause.

!!! ERROR [70] Nonnumeric item in arithmetic expression

Used alphanumeric item in an Arithmetic Expression clause.

!!! ERROR [71] The table containing literals, AS formats and headings has overflowed

The literal table overflowed its maximum size of 5,915 words.

!!! ERROR [72] Invalid occurrence number

Attempted to subscript past the end of a table.

!!! ERROR [73] Too many or too few parameters

Wrong number of parameters for JULIAN-DATE, TIMESTAMP-TIME, or TIMESTAMP-DATE clauses.

!!! ERROR [74] Too many PARAM declarations

Number of parameters for the current ENFORM session exceeds 32. Clear some with the CLOSE statement.

!!! ERROR [75] Invalid occurrence specification. Not in range [1,64]

User-defined table cannot contain more than 64 elements.

```
!!! ERROR [76] Variable subscript illegal in this context
```

Subscript used must be an explicit numeric literal, not a field name.

```
!!! ERROR [77] A destination name must be specified
```

An item in a FIND statement needs to be assigned to an output field name, because the name-correspondence rules are insufficient here.

```
!!! ERROR [78] The attribute UNIQUE may not be used with an OVER clause
```

UNIQUE may not be used with aggregates computed OVER a grouped-item.

```
!!! ERROR [79] TAB 0, SKIP 0 or FORM 0 not defined
```

Number must be greater than zero.

```
*** WARNING [80] Section name not found
```

Misspelled or nonexistent section in the Edit file.

```
!!! ERROR [81] The preceding text contains a syntactically incorrect element
```

Check the preceding line. If OK, check the next few preceding lines.

```
*** WARNING [82] Value is being truncated to one character
```

The value for the Option Variable must be a single ASCII character.

```
!!! ERROR [83] The type of the argument in the SET clause is invalid
```

Assigning a string to a numeric or vice versa or assigning a non-integer numeric literal.

```
!!! ERROR [84] Too many OVER clauses
```

Number of AFTER CHANGE, BEFORE CHANGE, TOTAL, SUBTOTAL, CUM, or PCT clauses in the LIST statement exceeds 64.

```
!!! ERROR [85] More than one PCT or CUM modifies list item
```

Only one PCT or CUM clause allowed per item.

```
!!! ERROR [86] Server QP process has failed repeatedly
```

Either the primary or the backup process for the ENFORM query processor has failed more than 10 times. The QP terminates abnormally when this condition occurs and must be restarted (preferably in another CPU).

```
*** WARNING [87] No RUN file has been named
```

Specify the Edit file name. No Edit file name has been specified in this session by a previous ?RUN or ?EDIT command.

```
!!! ERROR [88] Illegal CHECKPOINT parameter
```

The primary process for the ENFORM query processor executed a bad checkpoint call; probably an internal error. The QP terminates abnormally when this condition occurs and must be restarted.

```
!!! ERROR [89] Too many expressions in target list
```

A LIST or FIND statement contains too many Arithmetic Expression or Logical Expression clauses.

```
!!! ERROR [90] All field names referenced in a qualification
    aggregate must belong to the same record
```

All fields in the expression being aggregated, the over-item, and the embedded WHERE clause must belong to the same record.

```
*** WARNING [91] No report will be listed. The target list is composed of literals
    only
```

An ENFORM report will not print alphanumeric and/or numeric literals only. Include at least one field name from an opened file description.

```
!!! ERROR [92] At least one record has no LINK or a WHERE clause relating it to any
    other record
```

You have entered query specifications that reference two or more record descriptions. At least one of these record descriptions has no relationship (link) to any other record description in the query. ENFORM will not execute the query because a cross-product could result. Check your query specifications and add the necessary LINK statements or a WHERE clause.

```
!!! ERROR [93] A user aggregate having an end-expression may not be used in this
    context
```

A user aggregate declared with an end-expression cannot be used as a qualification aggregate OVER a grouped-item.

```
!!! ERROR [94] Your dictionary is bad
```

Refer to the *Data Definition Language (DDL) Programming Manual.*

```
!!! ERROR [95] Missing dictionary
```

Wrong subvolume or dictionary does not exist.

```
!!! ERROR [96] Invalid dictionary subvolume name
```

Internal error.

```
!!! ERROR [98] Insufficient memory available to OPEN record description
```

Internal error.

```
!!! ERROR [99] Multiply defined SECTION name
```

Section name may appear only once in a ?COMPILE, ?RUN, or ?SOURCE command.

```
!!! ERROR [100] Undefined SET variable
```

User-defined item or parameter used in a SET statement has not been defined yet.

```
*** WARNING [101] The param table would overflow if updated to the SET value
```

Parameter table is full and the last value has not been added. Use the CLOSE statement to clear parameter values not needed.

```
!!! ERROR [102] Field referenced in TITLE statement not found in target list
```

Usually an internal error with some unsupported item within a AFTER CHANGE, AT END, AT START, BEFORE CHANGE, FOOTING, SUBFOOTING, SUBTITLE, or TITLE statement or clause.

```
!!! ERROR [103] Invalid ENFORM version. Recompile to execute
```

The compiled physical file was compiled by a version of ENFORM which is not compatible with the current ENFORM version. Compile it again.

```
!!! ERROR [104] Output line would exceed buffer space.
```

Divide the output line in the LIST statement using SKIP or FORM clauses.

```
!!! ERROR [105] SUBTOTAL, TOTAL, CUM, and PCT only modify numericitems.
```

Cannot use alphanumeric string items. Numeric strings are allowed.

```
!!! ERROR [106] Field or expression must be numeric.
```

Self-explanatory.

```
!!! ERROR [107] Insufficient memory to build query processor representation of your
query
```

Reduce the size of the requested ENFORM query.

```
!!! ERROR [108] An aggregate may not be used in a SUPPRESS clause
```

Self-explanatory.

```
!!! ERROR [109] An aggregate may not be used as a parameter to a function
```

Self-explanatory.

```
!!! ERROR [110] Insufficient memory available to produce the report.
```

Try running ENFORM with the MEM option greater than 52. (Refer to the *GUARDIAN Operating System Programming Manual.*)

```
!!! ERROR [112] Illegal dictionary description (SERVER-related failure on name).
```

The dictionary description for an ENFORM server (process file) must specify a file type of UNSTRUCTURED or no file type at all.

```
!!! ERROR [113] An aggregate may not be used in this context with PCT
```

Only the aggregates SUM and COUNT can be used with PCT and they must be used alone (not in an expression).

```
!!! ERROR [114] Incorrect reply length (SERVER-related failure on name)
```

ENFORM server (process file) returned a reply with an unexpected length to the query processor. One way to get this error is to specify an odd data record byte size.

```
!!! ERROR [115] Dictionary is outdated. Recompile with D00 DDL or later
```

As of release T9102C10, ENFORM accepts only dictionaries compiled with DDL Version D00 or later. Current dictionary has an old version number; recompile it with a new version of DDL.

```
!!! ERROR [133] Invalid dictionary specification
```

The dictionary name specified in either a DICTIONARY statement or a ?DICTIONARY command is not a valid file name.

```
!!! ERROR [136] Page count exceeded
```

Your report has produced more pages than the number specified in the @PAGES option.

```
!!! ERROR [137] Invalid date specified
```

The data being formatted with an AS DATE clause is not a valid date.

```
!!! ERROR [138] Invalid time specified
```

The data being formatted with an AS TIME clause is not a valid time.

```
!!! ERROR [143] Data type not supported
```

The data type of a field in the DDL record description is not supported by ENFORM. (For example, ENFORM does not support COMPLEX, BINARY, or LOGICAL data types.)

```
!!! ERROR [166] String literal must be terminated with a quotation mark
```

Closing quotation mark is missing. Remember that a string literal cannot be continued from one source line to the next.

```
!!! ERROR [167] String literal cannot contain more than 127 characters
```

Self-explanatory.

```
!!! ERROR [168] TOTAL may not be specified OVER a BY item
```

TOTAL can only be specified OVER ALL. If you wish to compute a total over a BY item, specify SUBTOTAL instead.

```
!!! ERROR [169] ?RUN command is ignored unless entered interactively
```

The ?RUN command must be typed in at the terminal.

```
!!! ERROR [170] Illegal value for this option variable
```

Check the syntax of the Option Variable clauses in the *ENFORM Reference Manual* for the values allowed.

```
!!! ERROR [172] Item on left side of assign operator must be a field in the FIND record
```

The *output-field-name* in a FIND statement cannot be a field from an input record or the name of the FIND record itself.

```
!!! ERROR [173] Value must be a single ASCII character, not "^" or "-"
```

This is a restriction on the value for the Option Variable @NEWLINE.

```
!!! ERROR [174] Value is being truncated to 15 characters
```

The value for this Option Variable must be a string literal containing 15 characters or less.

```
!!! ERROR [175] A subscript range cannot be used in a field in a FIND statement
```

Specify each item in the range individually.

```
!!! ERROR [176] Help item phrase must be less than 32 characters long.
```

The phrase following the ?HELP keyword must be less than 32 characters long, including embedded blanks and the initial question mark (if present).

```
!!! ERROR [177] Parameter is treated like a literal here. Its value cannot be changed.
```

In certain cases, ENFORM treats a parameter exactly like a numeric literal. This means that you cannot change the value of the parameter at execution time, either with a Command Interpreter PARAM command or an ENFORM SET statement. Refer to the PARAM statement in the *ENFORM Reference Manual* for more details.

```
!!! ERROR [178] Record on right of link optional is linked back to record on left
```

Within your query specifications, a link exists that illegally links the record description specified on the right side of a LINK OPTIONAL statement back to the record description specified on the left side of the LINK OPTIONAL statement. Refer to the *ENFORM Reference Manual* for more information about the rules involving the LINK OPTIONAL statement.

```
!!! ERROR [179] Record appears on the right side of more than one link optional
```

Your query specifications contain one or more illegal links that violate rule 2 for the LINK OPTIONAL statement. (Rule 2 states that a record description can appear only once on the right side of a LINK OPTIONAL statement. Refer to the *ENFORM Reference Manual* for information about the rules involving the LINK OPTIONAL statement.

```
!!! ERROR (Attempt to divide by zero)
```

An error occurred because an attempt was made to divide by a field containing a data value of zero.

```
!!! ERROR (Cost tolerance exceeded): required cost = n
```

The strategy that the query processor will need to use to execute your query is greater than the value (*n*) you specified for the @COST-TOLERANCE optional variable.

```
!!! ERROR (Messages not in expected order)
```

The query processor received an unexpected message from the query compiler/report writer. If the query processor is running as a server in NonStop mode, this message indicates a failure in the primary process. Otherwise, this message indicates an error in the ENFORM software.

```
!!! ERROR (SORT failure) sort-error-number [ *** FILE ERROR file-error-number ]
[ on name ]
```

An error occurred in the SORT process. For an explanation of the *sort-error-number*, refer to the *Sort/Merge User's Guide*

```
Invalid response from the Query Processor
```

The query compiler/report writer process has received an invalid response from the query processor. This message usually indicates an error in the ENFORM software. Report this message to your system manager.

*** FILE ERROR TYPE MESSAGES

File management errors are reported through ENFORM with *** *FILE ERROR...* messages. In the messages below, *#file-error-number* is a GUARDIAN file system error number. *name* is the physical file name.

```
*** FILE ERROR (Abnormal termination of Query Processor)
```

Self-explanatory.

```
*** FILE ERROR (Communication with Query Processor failed) #file-error-number on name
```

ENFORM lost communication with the query processor.

```
*** FILE ERROR (CONTROL failure) #file-error-number on name
```

A control failure occurred on the physical file named.

```
*** FILE ERROR (CREATE failure) #file-error-number on name
```

There was a problem creating the physical file.

```
*** FILE ERROR (Dictionary file access failure) #file-error-number on name
```

Refer to the *Data Definition Language (DDL) Programming Manual.*

```
*** FILE ERROR (Illegal ENFORM execution file) on name
```

The file must be a compiled query file created with the ?COMPILE command.

```
*** FILE ERROR (Illegal list device) on name
```

Listing device name is misspelled or does not exist.

```
*** FILE ERROR (Illegal input device) on name
```

The input file name or Edit file name is misspelled or does not exist.

```
*** FILE ERROR (Not an Edit file) on name
```

File named is not an Edit file.

```
*** FILE ERROR (OPEN failure) #file-error-number on name
```

There was a problem opening the physical file.

```
*** FILE ERROR (POSITION failure) #file-error-number on name
```

A position failure occurred on the physical file named.

```
*** FILE ERROR (Process nonexistent, insufficient system resources or full queue)
    #file-error-number on name
```

The server query processor named in the ?ATTACH command does not exist or cannot accept more users.

```
*** FILE ERROR (PURGE failure) on name
```

There was a problem purging the physical file.

*** FILE ERROR (READ failure) #file-error-number on name

ENFORM could not read the physical file named.

*** FILE ERROR (RENAME failure) on name

There was a problem renaming the physical file.

*** FILE ERROR (SERVER-related failure) # number on name

There was a failure related to the use of an ENFORM server (process file). The number and name are optional values supplied by the server instead of a standard GUARDIAN file error and file name.

*** FILE ERROR (SETMODE failure) #file-error-number on name

A SETMODE error occurred on the physical file named.

*** FILE ERROR (Specified ENFORM compile file exists as edit or TAL object file) on name

The file must be a compiled query file created with an ENFORM ?COMPILE command.

*** FILE ERROR (Unable to open ENFORM message table #file-error-number on name

Self-explanatory. ENFORM terminates abnormally. Correct the problem with the message table and restart the session.

*** FILE ERROR (Unable to position ENFORM message table) #file-error-number on name

ENFORM is unable to use the message table file. The session continues but all messages contain "???" instead of text.

*** FILE ERROR (Unable to read ENFORM message table) #file-error-number on name

ENFORM is unable to use the message table file. The session continues but all messages contain "???" instead of text.

*** FILE ERROR (WRITE failure) #file-error-number on name

A write error occurred on the physical file named. If error 45 appears on the target file (where *name* appears as #nnnn), the target file has overflowed at least twice. See Section 5 for information about controlling the size of the target file.

ENFORM TRAP MESSAGES

ENFORM TRAP: nnn S: xxxxxx P: xxxxxx E: xxxxxx L: xxxxxx

The ENFORM Compiler/Report Writer process has failed. *nnn* is the trap number as described in the *GUARDIAN Operating System Programming Manual. xxxxxx* are values in the hardware registers.

ENFORM QP TRAP: nnn S: xxxxxx P: xxxxxx E: xxxxxx L: xxxxxx

The ENFORM Query Processor process has failed. *nnn* is the trap number as described in the *GUARDIAN Operating System Programming Manual. xxxxxx* are values in the hardware registers.

BUILDMK ERROR MESSAGES

*** ERROR A ? may only appear as the first char in the first word of a HELP phrase

A question mark can only appear as the first character in the first word of the phase that identifies a HELP section or subsection.

*** ERROR Edit file contains ?HELP section but no ?MESSAGES section

The Edit file version of the message table must contain a ?MESSAGES section if a ?HELP section is included.

*** ERROR Edit file name parameter is missing

The Edit file name parameter of the BUILDMK command is missing. Re-issue the BUILDMK command and include the Edit file name parameter.

*** ERROR Edit file contains ?MESSAGES section but no ?HELP section

The Edit file version of the message table must contain a ?HELP section if a ?MESSAGES section is included.

*** ERROR Edit file must not be empty

The Edit file specified on the BUILDMK command is empty. Specify the correct Edit file and re-issue the BUILDMK command.

*** ERROR First parameter in command line must be an edit file name

The first parameter specified for the BUILDMK command must be the name of an Edit file containing the Edit version of the message table.

*** ERROR Identifier contains an illegal character

An identifier specified in the Edit file version of the message table contains an illegal character.

*** ERROR Identifier must begin with an alphabetic character or ^

An identifier specified in the Edit file version of the message table must begin with either an alphabetic character or a circumflex.

*** ERROR Identifier must contain less than 32 characters.

Self-explanatory.

*** ERROR Identifier must not end with a hyphen

An identifer specified in the Edit file version of the message table must not end with a hyphen.

*** ERROR In a reserved word redefinition, the new reserved word is missing

Supply the new reserved word.

*** ERROR In a reserved word redefinition, the old reserved word is missing

Supply the old reserved word.

*** ERROR Invalid edit file name was specified

An invalid Edit file name was specified in the BUILDMK command. Correct the Edit file name and reissue the BUILDMK command.

*** ERROR Invalid key-sequenced file name was specified

An invalid name was specified for the key-sequenced file parameter of the BUILDMK command. Correct the key-sequenced file name and reissue the BUILDMK command.

*** ERROR Key-sequenced file must be empty.

The key-sequenced file that is to contain the message table must be empty before you run BUILDMK.

*** ERROR Key-sequenced file name parameter is missing

The name of the key-sequenced file is missing from the BUILDMK command. Reissue the BUILDMK command and include the name of the key-sequenced file.

*** ERROR Primary key for key-sequenc ed file must be at offset 0 and have length 34

Self-explanatory.

*** ERROR Second parameter in command line must be a key-sequenced file name

The second parameter of the BUILDMK command must be the name of a key-sequenced file.

*** ERROR ?HELP must appear in columns 1-5 and all other columns must be blank

The characters ?HELP must appear in columns 1 thru 5 of the first line of the ?HELP section. All other columns on this line must contain blanks.

*** ERROR ?HELP subsection must contain at least one subsection

Self-explanatory.

*** ERROR ?MESSAGES must appear in columns 1-9 an d all other columns must be blank

The characters ?MESSAGES must appear in columns 1 thru 9 of the first line of the ?MESSAGES section. All other columns on this line must contain blanks.

*** ERROR ?MESSAGES section must contain at least one line of text

Self-explanatory.

*** ERROR ?VOCABULARY must appear in columns 1-11 a nd all other columns must be blank

The characters ?VOCABULARY must appear in columns 1 thru 11 of the first line of the ?VOCABULARY section. All other columns on this line must contain blanks.

*** ERROR ?VOCABULARY section must redefine at least one reserved word

Self-explanatory.

*** FILE ERROR (EDITREAD read error) on name

The indicated error occurred on the specified file during the execution of BUILDMK.

*** FILE ERROR (EDITREAD sequence error) on name

The indicated error occurred on the specified file during the execution of BUILDMK.

*** FILE ERROR (EDITREAD text file format error) on name

The indicated error occurred on the specified file during the execution of BUILDMK.

*** FILE ERROR (EDITREADINIT I/O error) on name

The indicated error occurred on the specified file during the execution of BUILDMK.

*** FILE ERROR (OPEN failure) #file-error-number on name

The indicated error occurred on the specified file during the execution of BUILDMK.

*** FILE ERROR (POSITION failure) #file-error-number on name

The indicated error occurred on the specified file during the execution of BUILDMK.

*** FILE ERROR (READ failure) #file-error-number on name

The indicated error occurred on the specified file during the execution of BUILDMK.

*** FILE ERROR (WRITE failure) #file-error-number on name

The indicated error occurred on the specified file during the execution of BUILDMK.

APPENDIX C

SAMPLE DATA BASE

All of the examples in this guide use the sample relational data base in this appendix. Refer to the *Data Definition Language (DDL) Programming Manual* for a more complete description of the structure of a data base. This sample data base has two subsections: order entry records and employee records. The order entry records contain information on the parts in stock and orders for parts. The employee records contain information on the personnel and the organization of the company.

The names of the sample data base records and their contents are as follows:

- Order entry records

order	—	invoice for parts
customer	—	customer information
fromsup	—	purchase orders for parts from supplier
odetail	—	detailed invoice for parts
parts	—	inventory of parts in stock
supplier	—	supplier information

- Employee records

region	—	regional information
branch	—	branch information
employee	—	employee information

Each file has designated key fields on which different files may be linked. Figure C-1 indicates the relationships between the files by arrows.

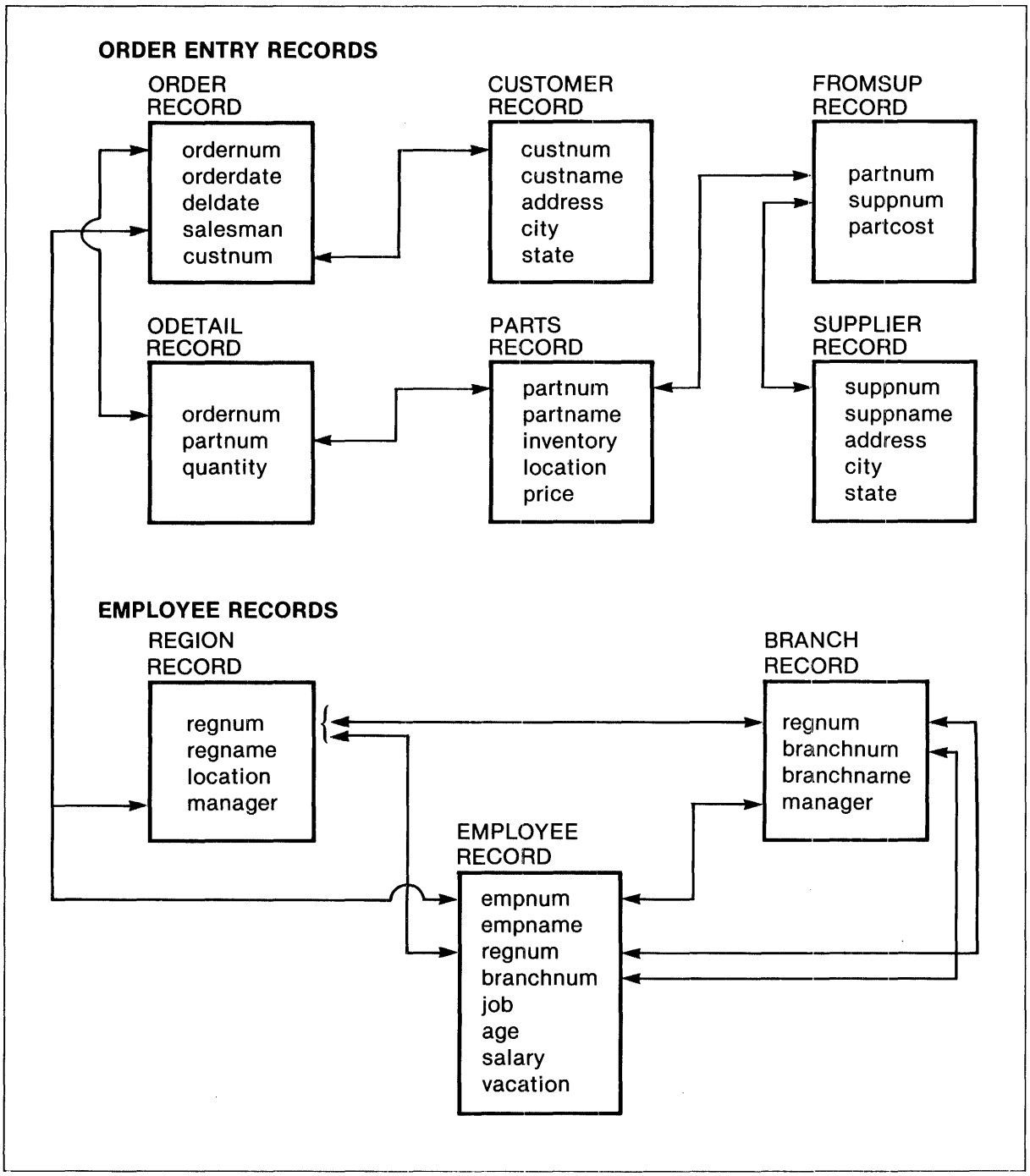


Figure C-1. Diagram of Sample Relational Data Base

The dictionary for the sample data base describes the records. It serves as a template to ENFORM for retrieval of records. Figure C-2 is a source listing of the dictionary.

```
?DICT !
?FUP ddlfup!
  DEF partnum PIC "9(4)" HEADING "Part/Number".
  DEF empnum PIC "9(4)".
  DEF custnum PIC "9(4)".
*====================  order entry records  ======================
  RECORD order.
         FILE IS "$mkt.sample.order"        KEY-SEQUENCED.
    02 ordernum;          PIC "999".
    02 orderdate.
       05 omonth;         PIC "99".
       05 oday;           PIC "99".
       05 oyear;          PIC "99".
    02 deldate.
       05 dmonth;         PIC "99".
       05 dday;           PIC "99".
       05 dyear;          PIC "99".
    02 salesman;          TYPE empnum.
    02 custnum,           TYPE *.
  KEY IS ordernum.
  KEY "sn" IS salesman.
  KEY "cn" IS custnum.
  END

  RECORD customer.
         FILE IS "$mtk.sample.customer"     KEY-SEQUENCED.
    02 custnum;           TYPE *.
    02 custname;          PIC "X(18)".
    02 address;           PIC "X(22)".
    02 city;              PIC "X(14)".
    02 state;             PIC "X(12)".
  KEY IS custnum.
  KEY "cn" IS custnum.
  END

  RECORD fromsup.
         FILE IS "$mkt.sample.fromsup"      KEY-SEQUENCED.
    02 primkey.
       05 partnum;        TYPE *.
       05 suppnum;        PIC "999".
    02 partcost;          PIC "999999V99";   DISPLAY "M<ZZZ,ZZ9.99>".
  KEY IS primkey.
  END
```

Figure C-2. Dictionary Source Listing of Sample Relational Data Base

```
RECORD odetail.
       FILE IS "$mkt.sample.odetail"        KEY-SEQUENCED.
  02 primkey.
     05 ordernum;      PIC "999".
     05 partnum;       TYPE *.
  02 quantity;         PIC "999".
KEY IS primkey.
END

RECORD parts.
       FILE IS "$mkt.sample.parts"          KEY-SEQUENCED.
  02 partnum;          TYPE *.
  02 partname;         PIC "X(18)".
  02 inventory;        PIC "999S".
  02 location;         PIC "XXX".
  02 price;            PIC "999999V99".
KEY IS partnum.
KEY "pn" IS partname.
END

RECORD supplier.
       FILE IS "$mkt.sample.supplier"       KEY-SEQUENCED.
  02 suppnum;          PIC "999".
  02 suppname;         PIC "X(18)".
  02 address;          PIC "X(22)".
  02 city;             PIC "X(14)".
  02 state;            PIC "X(12)".
KEY IS suppnum.
KEY "su" IS suppname.
END
```

Figure C-2. Dictionary Source Listing of Sample Relational Data Base (Continued)

```
*====================== employee records ==========================
  RECORD region.
         FILE IS "$mkt.sample.region"          KEY-SEQUENCED.
    02 regnum;              PIC "99".
    02 regname;             PIC "X(12)".
    02 location;            PIC "X(14)".
    02 manager;             PIC "9999".
  KEY IS regnum.
  KEY "rn" IS regname.
  END

  RECORD branch.
         FILE IS "$mkt.sample.branch"          KEY-SEQUENCED.
    02 primkey.
       05 regnum;           PIC "99".
       05 branchnum;        PIC "99".
    02 branchname;          PIC "X(14)".
    02 manager;             TYPE empnum.
  KEY IS primkey.
  END

  RECORD employee.
         FILE IS "$mkt.sample.employee"        KEY-SEQUENCED.
    02 empnum;              TYPE *.
    02 empname;             PIC "X(18)".
    02 dept.
       05 regnum;           PIC "99".
       05 branchnum;        PIC "99".
    02 job;                 PIC "X(12)".
    02 age;                 PIC "99".
    02 salary;              PIC "999999".
    02 vacation;            PIC "99".
  KEY IS empnum.
  KEY "en" IS empname.
  KEY "dp" IS dept.
  END
```

Figure C-2. Dictionary Source Listing of Sample Relational Data Base (Continued)

Figure C-3 shows the record occurrences associated with each of the record types.

ORDER Records:

ORDERNUM	OMONTH	ODAY	OYEAR	DMONTH	DDAY	DYEAR	SALESMAN	CUSTNUM
21	1	10	78	4	10	78	205	1234
25	1	23	78	6	15	78	212	7777
30	2	6	78	7	1	78	222	926
32	2	17	78	7	20	78	204	21
35	3	17	78	8	10	78	231	543
38	3	19	78	8	20	78	218	123
41	3	27	78	9	1	78	207	7654
45	4	20	78	9	15	78	212	324
48	5	12	78	10	10	78	225	3333
51	6	1	78	10	20	78	210	143
66	7	9	78	11	1	78	205	3210
122	7	21	78	12	15	78	221	5635

CUSTOMER Records:

CUSTNUM	CUSTNAME	ADDRESS	CITY	STATE
21	CENTRAL UNIVERSITY	UNIVERSITY WAY	PHILADELPHIA	PENN
123	BROWN MEDICAL CO	100 CALIFORNIA STREET	SAN FRANCISCO	CALIFORNIA
143	STEVENS SUPPLY	2020 HARRIS STREET	DENVER	COLORADO
324	PREMIER INSURANCE	3300 WARBASH	LUBBOCK	TEXAS
543	FRESNO STATE BANK	2300 BROWN BLVD	FRESNO	CALIFORNIA
926	METTALL-AG.	12 WAGNERRING	FRANKFURT	GERMANY
1234	DATASPEED	300 SAN GABRIEL WAY	NEW YORK	NEW YORK
3210	BESTFOODS MARKET	3333 PLELPS STREET	LINCOLN	NEBRASKA
3333	DEUTSCHE STAHL	SIEMENS-STRASSE	DUISBERG	GERMANY
5635	VEREINIGTE CHEMIE	45 FRANKENSTRASSE	MUENCHEN	GERMANY
7654	MOTOR DISTRIBUTING	2345 FIRST STREET	CHICAGO	ILLINOIS
7777	SLEEPWELL HOTELS	9000 PETERS AVENUE	DALLAS	TEXAS

Figure C-3. Listing of Records in Sample Relational Data Base

FROMSUP Records:

Part Number	SUPPNUM	PARTCOST
212	1	92000.00
244	1	87000.00
1403	1	22000.00
2001	1	1500.00
2002	1	1000.00
...	...	...
6401	2	1200.00
6401	3	1100.00
6402	2	1100.00
6402	3	2600.00
6603	2	2600.00
7102	10	6000.00
7301	1	2400.00

ODETAIL Records:

ORDERNUM	Part Number	QUANTITY
21	244	1
21	2001	2
21	2403	2
21	4103	2
25	244	1
25	5103	1
25	6301	2
25	6402	10
...	...	...
122	3103	20
122	3201	3
122	4103	40
122	5103	3
122	7102	7
122	7301	8

Figure C-3. Listing of Records in Sample Relational Data Base (Continued)

PARTS Records:

Part Number	PARTNAME	INVENTORY	LOCATION	PRICE
212	SYSTEM 192KB CORE	7	J87	92000.00
244	SYSTEM 192KB SEMI	3	B78	87000.00
1403	PROC 96KB SEMI	21	A21	22000.00
2001	DECIMAL ARITH	-100	X10	1500.00
2002	ENSCRIBE MICRO	200	X11	1000.00
...	...	...	...	...
5502	LP 300 LPM	6	L98	11500.00
5504	LP 900 LPM	-1	L88	21000.00
5505	LP 1500 LPM	0	L78	42000.00
6201	SYNC CONTROLLER	-16	A34	5800.00
6301	ASYNC CONTROLLER	21	A35	2900.00
6302	ASYNC EXTENSION	34	A36	4300.00
6401	TERM CRT CH AR	54	V67	1500.00
6402	TERM CRT PAGE	-32	V68	1500.00
6603	TERM HARD COPY	40	V66	3200.00
7102	CABINET LARGE	20	F76	68000.05
7301	POWER MODULE	32	H76	2400.06

SUPPLIER Records:

SUPPNUM	SUPPNAME	ADDRESS	CITY	STATE
1	TANDEM COMPUTERS	19333 VALLCO PARKWAY	CUPERTINO	CALIFORNIA
2	DATA TERMINAL	2000 BAKER STREET	IRVINE	CALIFORNIA
3	DISPLAY INC	7600 EMERSON	PALO ALTO	CALIFORNIA
6	INFOMATION STORAGE	1000 INDUSTRY DRIVE	LEXINGTON	MASS
8	MAGNETICS CORP	7777 FOUNTAIN WAY	SEATTLE	WASHINGTON
10	STEELWORK INC	6000 LINCOLN LANE	SUNNYVALE	CALIFORNIA
15	DATADRIVE	100 MACARTHUR	DALLAS	TEXAS

Figure C-3. Listing of Records in Sample Relational Data Base (Continued)

REGION Records:

REGNUM	REGNAME	LOCATION	MANAGER
1	EAST	NEW YORK	29
2	CENTRAL	CHICAGO	104
3	WEST	DALLAS	72
4	CANADA	TORONTO	343
5	GERMANY	FRANKFURT	43
6	ENGLAND	LONDON	87
99	HEADQUARTERS	CUPERTINO	1

BRANCH Records:

REGNUM	BRANCHNUM	BRANCHNAME	MANAGER
1	1	NEW YORK	75
1	2	NEW JERSEY	129
2	1	CHICAGO	23
2	2	HOUSTON	109
2	3	ST.LOUIS	111
3	1	DALLAS	321
3	2	LOS ANGELES	337
3	3	SAN FRANCISCO	89
4	1	TORONTO	178
4	2	VANCOVER	93
5	1	FRANKFURT	180
5	2	DUESSELDORF	39
5	3	MUENCHEN	32
6	1	LONDON	65
99	1	CUPERTINO	88

Figure C-3. Listing of Records in Sample Relational Data Base (Continued)

EMPLOYEE Records:

EMPNUM	EMPNAME	REGNUM	BRANCHNUM	JOB	AGE	SALARY	VACATION
1	ROGER GREEN	99	1	MANAGER	37	39500	2
23	JERRY HOWARD	2	1	MANAGER	34	37000	10
29	JACK RAYMOND	1	1	MANAGER	39	36000	1
32	THOMAS RUDLOFF	5	3	MANAGER	43	38000	0
...	...	...	...	...	...	...	...
201	JIM HERMAN	1	1	SALESMAN	27	19000	13
202	LARRY CLARK	1	1	SYS.-ANAL.	30	25000	7
203	KATHRYN DAY	1	1	SECRETARY	24	12000	12
204	TOM HALL	1	1	SALESMAN	35	26000	0
205	GEORGE FORSTER	1	2	SALESMAN	39	30000	4
206	DAVE FISHER	2	1	SALESMAN	32	25000	7
207	MARK FOLEY	2	1	SALESMAN	27	23000	10
...	...	...	...	...	...	...	...

Figure C-3. Listing of Records in Sample Relational Data Base (Continued)

APPENDIX D

EXAMPLE ENFORM PROGRAMS

This appendix contains complete programs illustrating various features of the ENFORM language. All of the programs use the sample relational data base shown in Appendix C.

The following example illustrates two ways of specifying a linking relationship between record descriptions. The first program contains a LINK statement; the second program contains a WHERE clause. They produce identical reports. Notice that *partnum* requires qualification because it appears in both *odetail* and *parts*.

The BY clause groups the records on *ordernum* first, then on *partnum*.

The AS clause uses the mask format for *inventory* and *price*. The format for *inventory* uses the decorations feature to print OUT for out of stock and to print blanks for in stock.

The WHERE clause restricts the report to records where *ordernum* is less than 30.

```
?DICTIONARY $mkt.dictry
OPEN odetail, parts;
LINK odetail TO parts VIA partnum
LIST BY ordernum,
     BY odetail.partnum,
     partname,
     inventory AS "[MA1'OUT ',PA1'    '] M<ZZZZZ9>",
     price AS M<ZZZZZZ.>,
  WHERE ordernum LT 30;
```

```
?DICTIONARY $mkt.dictry
OPEN odetail, parts;
LIST BY ordernum,
     BY parts.partnum,
     partname,
     inventory AS "[MA1'OUT ',PA1'    '] M<ZZZZZ9>",
     price AS M<ZZZZZZ.>,
WHERE odetail.partnum EQUAL parts.partnum
AND ordernum LT  30;
```

Report:

```
            Part
ORDERNUM  Number       PARTNAME        INVENTORY   PRICE
--------  ------  ------------------   ---------  -------

      21     244  SYSTEM 192KB SEMI            3   87000.
            2001  DECIMAL ARITH          OUT 100    1500.
            2403  MEM MOD 96K MOS             12    9600.
            4103  DISK 160MB                   7   24500.
      25     244  SYSTEM 192KB SEMI            3   87000.
            5103  MAG TAPE DR 8/16             8    8000.
            6301  ASYNC CONTROLLER       OUT  21    2900.
            6402  TERM CRT PAGE          OUT  32    1500.
```

The following program uses the CENTER clause to center *ordernum*. Notice that the formatted width "999", as specified in the dictionary, is used for centering instead of the value of a field for the report line.

The AS clause formats *orderdate*, using the mask format and symbol substitution modifier to substitute X for the 9 symbol.

```
?DICTIONARY $mkt.dictry
OPEN order;
LIST ordernum CENTER,
     orderdate AS "[SS'9X'] M<ZX XX, 19XX>";
```

Report:

```
ORDERNUM   ORDERDATE
--------  -----------

    21     1 10, 1978
    25     1 23, 1978
    30     2 06, 1978
    32     2 17, 1978
    35     3 03, 1978
    38     3 19, 1978
    41     3 27, 1978
    45     4 20, 1978
    51     6 01, 1978
    66     7 09, 1978
   122     7 21, 1978
```

The following program resets the Option Variable @SUBTOTAL-LABEL. The new value appears in place of the default value asterisk (*).

Three record descriptions are linked, creating new logical record occurrences. The BY clause groups the records on the value of *ordernum*. Notice that *ordernum* requires qualification, because it appears in both *parts* and *odetail.*

The HEADING clause overrides the default *quantity* heading, creating a narrower column width in the report.

A new report item is created with the arithmetic expression (price * quantity). A subtotal is printed for the new item, assuming OVER *order.ordernum.*

The CUM OVER clause prints the cumulative values of the arithmetic expression for each group.

The WHERE clause restricts the records that contribute to the report.

```
?DICTIONARY $mkt.dictry
SET @SUBTOTAL-LABEL TO "SUBTOTAL";
OPEN order, odetail, parts;
LINK order TO odetail VIA ordernum,
     parts TO odetail VIA partnum;
LIST BY order.ordernum,
     parts.partnum AS M<9 999>,
     quantity HEADING "QTY",
     price,
     (price * quantity) AS M<Z,ZZZ,999.>
       HEADING "PRICE * QTY", SUBTOTAL, TOTAL,
     (price * quantity)
       CUM OVER order.ordernum  AS M<Z,ZZZ,999.>
       HEADING "ACCUMULATION/BY ORDER",
  WHERE order.ordernum > 60;
```

Report:

ORDERNUM	Part Number	QTY	PRICE	PRICE * QTY	ACCUMULATION BY ORDER
66	0 244	1	87000.00	87,000.	87,000.
	1 403	3	22000.00	66,000.	153,000.
	2 001	5	1500.00	7,500.	160,500.
	2 403	8	9600.00	76,800.	237,300.
	3 102	3	4800.00	14,400.	251,700.
	3 302	3	2800.00	8,400.	260,100.
	4 101	1	8000.00	8,000.	268,100.
	4 102	6	14500.00	87,000.	355,100.
	4 103	2	24500.00	49,000.	404,100.
	5 101	1	7400.00	7,400.	411,500.
	5 502	8	11500.00	92,000.	503,500.
	5 504	3	21000.00	63,000.	566,500.
	6 201	1	5800.00	5,800.	572,300.
	6 301	4	2900.00	11,600.	583,900.
	6 302	5	4300.00	21,500.	605,400.
	6 401	6	1500.00	9,000.	614,400.
	6 402	22	1500.00	33,000.	647,400.
	7 102	1	68000.05	68,000.	715,400.
	7 301	2	2400.06	4,800.	720,200.

SUBTOTAL				720,200.	
122	1 403	10	22000.00	220,000.	220,000.
	2 002	10	1000.00	10,000.	230,000.
	2 403	30	9600.00	288,000.	518,000.
	3 103	20	10500.00	210,000.	728,000.
	3 201	3	4800.00	14,400.	742,400.
	4 103	40	24500.00	980,000.	1,722,400.
	5 103	3	8000.00	24,000.	1,746,400.
	7 102	7	68000.05	476,000.	2,222,400.
	7 301	8	2400.06	19,200.	2,241,601.

SUBTOTAL				2,241,601.	

				2,961,801.	

The following program shows different ways of formatting dates. Notice that the JULIAN-DATE clause is used to convert the date to internal format before using the AS DATE clause. *Deldate* uses the default heading; the dates converted by the JULIAN-DATE clause use the HEADING clause. The CENTER clause centers the day of the week. The WHERE clause restricts the records that contribute to the report.

```
?DICTIONARY $mkt.dictry
OPEN order;
LIST custnum,
     deldate AS M<99/99/99>,
     JULIAN-DATE((1900 + dyear),dmonth,dday) AS DATE "DA3"
       HEADING "DAY OF WK" CENTER,
     JULIAN-DATE((1900 + dyear),dmonth,dday)
       AS DATE "MA DAO, Y4"
       HEADING "WRITTEN/DATE",
  WHERE custnum < 1000;
```

Report:

```
Customer                                        WRITTEN
 Number   DELDATE   DAY OF WK                     DATE
--------  --------  ---------  ------------------------------

      21  07/20/78     THU                 JULY TWENTIETH, 1978
     123  08/20/78     SUN               AUGUST TWENTIETH, 1978
     143  10/20/78     FRI             OCTOBER TWENTIETH, 1978
     324  09/15/78     FRI          SEPTEMBER FIFTEENTH, 1978
     543  08/10/78     THU                AUGUST TENTH, 1978
     926  07/01/78     SAT                  JULY FIRST, 1978
```

The following program resets the Option Variable @SUBTOTAL-LABEL. The new value appears in place of the default value asterisk (*).

Three record descriptions are linked together, creating a new logical record description. The BY clause groups the records on the value of *ordernum*. Notice that *ordernum* and *partnum* require qualification, because they appear in more than one record description.

The AS clause uses the mask format to print *partnum* in two columns. The HEADING clause overrides the default QUANTITY heading, creating a narrower column width in the report.

A new report item is created with the arithmetic expression (price * quantity). A subtotal is printed for the new report item, assuming OVER *order.ordernum*. Notice that asterisks are printed for the second subtotal, because the calculated value 2,241,600.83 exceeds the field width in the mask format. The mask format should be increased at least one more digit.

The WHERE clause restricts the records that contribute to the report.

The TITLE and SUBTITLE clauses print information at the top of the report. The subtitle is centered within the report. The AT START PRINT, AT END PRINT, BEFORE CHANGE, and AFTER CHANGE clauses print information within the report.

```
?DICTIONARY $mkt.dictry
SET @SUBTOTAL-LABEL TO "SUBTOTAL";
OPEN order, odetail, parts;
LINK order TO odetail VIA ordernum;
LINK parts TO odetail VIA partnum;
LIST BY order.ordernum,
     parts.partnum AS M<9 999>,
     quantity HEADING "QTY",
     price,
     (price * quantity) AS M<ZZZ,999.99>
       HEADING "PRICE * QTY", SUBTOTAL, TOTAL,
  WHERE order.ordernum > 51,
  TITLE "SUMMARY OF ORDERS" TAB 34 @DATE AS DATE * SKIP 2,
  SUBTITLE "RUN AT - " @TIME AS TIME * CENTER,
  AT START PRINT "BEGIN =========",
  AT END PRINT "END =========",
  BEFORE CHANGE ON order.ordernum PRINT "=== BEFORE CHANGE ===",
  AFTER CHANGE ON order.ordernum PRINT "=== AFTER CHANGE ===";
```

Report:

```
SUMMARY OF ORDERS                12/05/79

              RUN AT - 02:33:15  PM

BEGIN =========

            Part
ORDERNUM  Number  QTY     PRICE     PRICE * QTY
--------  ------  ---  ---------  -----------

=== AFTER CHANGE ===
      66   0 244    1   87000.00    87,000.00
           1 403    3   22000.00    66,000.00
           2 001    5    1500.00     7,500.00
           2 403    8    9600.00    76,800.00
           3 102    3    4800.00    14,400.00
           3 302    3    2800.00     8,400.00
         ... ...  ...        ...         ...

           6 402   22    1500.00    33,000.00
           7 102    1   68000.05    68,000.05
           7 301    2    2400.06     4,800.12
                                   -----------
SUBTOTAL                            720,200.17

=== BEFORE CHANGE ===
=== AFTER CHANGE ===
     122   1 403   10   22000.00   220,000.00
           2 002   10    1000.00    10,000.00
           2 403   30    9600.00   288,000.00
           3 103   20   10500.00   210,000.00
           3 201    3    4800.00    14,400.00
           4 103   40   24500.00   980,000.00
           5 103    3    8000.00    24,000.00
           7 102    7   68000.05   476,000.35
           7 301    8    2400.06    19,200.48
                                   -----------
SUBTOTAL                            **********

=== BEFORE CHANGE ===

                                   -----------
                                   -----------
                                    **********

 END =========
```

The following program creates two user variables, *invcount* and *counter*. These user variables are both assigned values in the LIST statement.

Three record descriptions are linked together, creating new logical records. The BY clause groups the records on the value of *ordernum*. Notice that *ordernum* and *partnum* require qualification, because they appear in more than one record description.

The AS clause uses the mask format to print *partnum* in two columns. The HEADING clause overrides the default QUANTITY and INVENTORY headings, creating narrower column widths for those fields.

The NOPRINT clause suppresses the printing of *invcount* and *counter*. These user variables are calculated and used in the IF/THEN/ ELSE expression. The IF/THEN/ELSE expression determines which value is be printed in the last column. The HEADING clause specifies the column title.

The WHERE clause restricts the records that contribute to the report.

```
?DICTIONARY $mkt.dictry
DECLARE invcount INTERNAL I14;
DECLARE counter INTERNAL I12;
OPEN order, odetail, parts;
LINK order TO odetail VIA ordernum,
     parts TO odetail VIA partnum;
LIST BY order.ordernum,
     parts.partnum AS M<9 999>,
     quantity HEADING "QTY",
     inventory HEADING "INV",
     invcount := (inventory + quantity) NOPRINT,
     counter := (invcount + 1) NOPRINT,
     (IF counter > 0 THEN counter ELSE BLANK) TOTAL
        HEADING "INV + QTY + 1/(OMIT NEGATIVES)",
  WHERE order.ordernum > 60;
```

Report:

```
            Part                          INV + QTY + 1
ORDERNUM    Number   QTY    INV    (OMIT NEGATIVES)
--------    ------   ---   -----   ----------------

      66    0 244      1       3                  5
            1 403      3      21                 25
            2 001      5    -100
            2 403      8      12                 21
            3 102      3      12                 16
          ... ...    ...     ...                ...
            6 201      1     -16
            6 301      4     -21
            6 302      5      34                 40
            6 401      6      54                 61
            6 402     22     -32
            7 102      1      20                 22
            7 301      2      32                 35
     122    1 403     10      21                 32
            2 002     10     200                211
            2 403     30      12                 43
            3 103     20      -4                 17
            3 201      3       6                 10
            4 103     40       7                 48
            5 103      3       8                 12
            7 102      7      20                 28
            7 301      8      32                 41
                                   ----------------
                                   ----------------
                                                748
```

The following program uses the AS clause mask format for *price* and *partnum*. The CENTER clause centers *inventory*. Notice that the formatted width 999S, as specified in the dictionary, is used for centering the values of *inventory* instead of the specific value of the field. The IF/THEN/ELSE clause determines which value is printed in the last column. The HEADING clause specifies the column title. The TOTAL clause sums up the last two columns.

The WHERE clause restricts records from contributing to the report when either of the conditions specified in its logical expression tests true.

The SUPPRESS clause limits the printing of records for all items out of stock, but does not limit these records from contributing to the totals. In this example the *inventory* total is negative.

```
?DICTIONARY $mkt.dictry
OPEN parts;
LIST location,
     price AS M<ZZZZZZ.>
     partnum AS M<9 999>,
     partname,
     inventory TOTAL CENTER,
     (IF inventory GE 0 THEN inventory ELSE BLANK) TOTAL
       HEADING "AVAIL/STOCK" AS M<ZZZ> CENTER,
  WHERE (partname BEGINS WITH "SY")
    OR (partname BEGINS WITH "LP"),
SUPPRESS inventory EQUAL -999 THRU 0;
```

Report:

```
                    Part                                      AVAIL
LOCATION   PRICE    Number       PARTNAME          INVENTORY  STOCK
--------  -------  ------  ------------------     ---------  -----

J87        92000.   0 212  SYSTEM 192KB CORE            7      7
B78        87000.   0 244  SYSTEM 192KB SEMI            3      3
L98        11500.   5 502  LP  300 LPM                 6      6
                                                    ---------  -----
                                                    ---------  -----
                                                         -1     16
```

The following program uses the HEADING clause to indicate the fourth column of the heading. The WHERE clause uses the pattern match logical expression to select only records with a blank value in column four of *branch*.

```
?DICTIONARY $mkt.dictry
OPEN branch;
LIST branchnum,
     branchname HEADING "BRANCH NAME/...*...........",
  WHERE (branchname EQUAL [ 3 " " -]);
```

Report:

```
              BRANCH NAME
 BRANCHNUM    ...*...........
 ---------    ---------------

         1    NEW YORK
         2    NEW JERSEY
         3    ST. LOUIS
         2    LOS ANGELES
         3    SAN FRANCISCO
```

The following program resets the Option Variable @SUBTOTAL-LABEL. The new value appears in place of the default value asterisk (*).

Three record descriptions are linked together, creating new logical records. The BY clause groups the records on the value of *ordernum*. Notice that *ordernum* and *partnum* require qualification, because they appear in more than one record description.

The AS clause uses the mask format to print *partnum* with a hyphen. The HEADING clause overrides the default QUANTITY heading, creating a narrower column width in the report.

A new report item is created with the arithmetic expression (price * quantity). A subtotal is printed for the new item, assuming OVER *order.ordernum*.

The MAX aggregate prints a single maximum value of the arithmetic expression for each group. The WHERE clause restricts the report to records where *ordernum* is between 50 and 70.

```
?DICTIONARY $mkt.dictry
SET @SUBTOTAL-LABEL TO "SUBTOTAL";
OPEN order, odetail, parts;
LINK order TO odetail VIA ordernum,
     parts TO odetail VIA partnum;
LIST BY order.ordernum,
     parts.partnum AS M<9-999>,
     quantity HEADING "QTY",
     price,
     (price * quantity) AS M<ZZ,ZZZ,999.>
       HEADING "PRICE * QTY", SUBTOTAL, TOTAL,
     MAX (price * quantity OVER order.ordernum)
       AS M<ZZZ,999.>
       HEADING "MAXIMUM/PRICE * QTY",
  WHERE order.ordernum EQUAL 50 THRU 70;
```

Report:

ORDERNUM	Part Number	QTY	PRICE	PRICE * QTY	MAXIMUM PRICE * QTY
51	1-403	4	22000.00	88,000.	269,500.
	2-001	4	1500.00	6,000.	
	2-002	4	1000.00	4,000.	
	2-003	4	500.00	2,000.	
	2-403	16	9600.00	153,600.	
	3-103	5	10500.00	52,500.	
	3-302	1	2800.00	2,800.	
	4-103	11	24500.00	269,500.	
	5-103	1	8000.00	8,000.	
	5-505	1	42000.00	42,000.	
	6-301	2	2900.00	5,800.	
	6-302	2	4300.00	8,600.	
	6-402	8	1500.00	12,000.	
	7-102	1	68000.05	68,000.	

SUBTOTAL				722,800.	
66	0-244	1	87000.00	87,000.	92,000.
	1-403	3	22000.00	66,000.	
	2-001	5	1500.00	7,500.	
	...	...	...	...	
	7-301	2	2400.06	4,800.	

SUBTOTAL				720,200.	

				1,443,000.	

The following program resets the Option Variable @SUBTOTAL-LABEL. The new value appears in place of the default asterisk (*).

Two user variables, *sumval* and *cntval*, are declared. They are both assigned values in the LIST statement. The *sumval* user variable uses a mask format in the DECLARE statement to assign a default display format. The *cntval* user variable uses an I format in the DECLARE statement to assign a default display format.

Three record descriptions are linked together, creating new logical records. The BY clause groups the records on *ordernum*. Notice that *ordernum*and *partnum* require qualification, because they appear in more than one record description.

The AS clause uses the mask format to print *partnum* with a hyphen. The HEADING clause overrides the default *quantity* heading, creating a narrower column width in the report.

A new report item is created with the arithmetic expression, (price * quantity). A subtotal is printed for the new item, assuming OVER *order.ordernum*.

The user variables, *sumval* and *cntval*, are assigned values. The NOPRINT clause suppresses the printing of these items.

The WHERE clause restricts the report to records where *ordernum* is over 60.

The AT END PRINT clause prints information at the end of the report. The SKIP clause indicates a new line.

```
?DICTIONARY $mkt.dictry
SET @SUBTOTAL-LABEL TO "SUBTOTAL";
OPEN order, odetail, parts;
DECLARE sumval AS M<ZZZ,ZZZ,999.99>;
DECLARE cntval AS I10;
LINK order TO odetail VIA ordernum,
LINK parts TO odetail VIA partnum;
LIST BY order.ordernum,
     custnum,
     parts.partnum AS M<9-999>,
     quantity HEADING "QTY",
     price,
     (price * quantity) AS M<ZZ,ZZZ,999.>
       HEADING "PRICE * QTY", SUBTOTAL, TOTAL,
     sumval := SUM ((price * quantity) OVER ALL) NOPRINT,
     cntval := COUNT (UNIQUE parts.partnum) NOPRINT,
  WHERE order.ordernum > 60,
  AT END PRINT TAB 30 "SUM OF ORDERS         = " sumval SKIP 1,
               TAB 30 "NO OF DIFFERENT PARTS = " cntval;
```

Report:

```
          Customer   Part
ORDERNUM   Number   Number QTY    PRICE    PRICE * QTY
--------  --------  ------ ---  ---------  -----------

      66      3210   0-244,  1   87000.00      87,000.
              3210   1-403,  3   22000.00      66,000.
              3210   2-001,  5    1500.00       7,500.
              3210   2-403,  8    9600.00      76,800.
               ...     ... ...       ...         ...
              3210   7-301,  2    2400.06       4,800.
                                              -----------
SUBTOTAL                                         720,200.

     122      5635   1-403, 10   22000.00     220,000.
              5635   2-002, 10    1000.00      10,000.
              5635   2-403, 30    9600.00     288,000.
               ...     ... ...       ...         ...
              5635   5-103,  3    8000.00      24,000.
              5635   7-102,  7   68000.05     476,000.
              5635   7-301,  8    2400.06      19,200.
                                              -----------
SUBTOTAL                                       2,241,601.

                                              -----------
                                              -----------
                                               2,961,801.


                  SUM OF ORDERS         =   2,961,801.00
                  NO OF DIFFERENT PARTS =          23
```

The following program uses a FIND statement to create a new output file with three fields. The fields are *regnum*, *regname*, and *avgsal* for all employees in each region.

Two record descriptions are linked together, creating new logical records. The BY clause groups the records on the value of *regnum*. Notice that *regnum* requires qualification, because it appears in both *region* and *employee*.

```
?DICTIONARY $mkt.dictry
OPEN, region, employee, region2;
LINK region.manager TO employee.empnum;
FIND region2
     (BY region.regnum,
         region.regname,
      avgsal := AVG(region.salary OVER region.regnum));
```

APPENDIX E

CHANGING THE MESSAGE TABLE TEXT

ENFORM retrieves error message text, help message text, and reserved word redefinitions from a special key-sequenced file, called the ENFORM message table. Tandem supplies a default version of the message table in ENFORMMK, a key-sequenced file that contains the ENFORM message and help text. ENFORM allows you to modify the message table so that you can:

- Redefine the ENFORM reserved words, system variable names, option variable names, and command names. For example, you can translate these words into a language other than English or create your own abbreviations.

- Change the content of the ENFORM warning, error, and informational messages. For example, you can add information to these messages or translate the messages into a language other than English.

- Modify the content of the help text by either changing the existing text or adding your own text. For example, you could add help text that describes the Data Definition Language (DDL).

HOW TO CHANGE THE MESSAGE TABLE

To change the message table, perform the following:

1. Create an Edit file version of the message table. When you create the Edit file version, the contents of the Edit file must conform to a prescribed format (described in detail later in this section). Briefly, you must place any redefinitions in a section called ?VOCABULARY, any message text in section called ?MESSAGES, and any help text in a section called ?HELP.

2 Convert the Edit file version of the message table into a key-sequenced version.

To simplify the process of modifying the message table, Tandem supplies two files in addition to ENFORMMK. These files are: ENFORMMT, an Edit file version of the default message table and BUILDMK, a file containing object code that converts the Edit version of the message table into the special key-sequenced file required by ENFORM. Consult your system manager for the name of the volume and subvolume on which ENFORMMK, ENFORMMT, and BUILDMK reside.

ENFORM allows you to create a new message table for either of the following purposes:

- To use as the new message table for the current ENFORM session. In this case, you identify the key-sequenced file containing the message table in the *message-table-filename* option of the ENFORM command.

- To replace the default message table. ENFORM uses the new message table for all ENFORM sessions. Note that replacing the default message table affects all individuals in the system who use ENFORM and do not specify their own message table.

The following paragraphs provide guidelines to be followed when creating a message table for the current ENFORM session or when creating a message table to replace the default message table.

Guidelines for Creating a Message Table for the Current Session

When you create a new message table for the current ENFORM session, ENFORM allows you to include only a ?VOCABULARY section (ENFORM uses your reserved words with the standard messages and help text), only ?MESSAGES and ?HELP sections (ENFORM uses your messages and help text with the standard reserved words), or ?VOCABULARY, ?MESSAGES, and ?HELP sections (ENFORM uses your message table and ignores the standard message table). To create and use the new message table, perform the following tasks:

1. Create a new Edit file version of the message table. Depending on your needs, you can either create an Edit file for your message text or use the File Utility Program (FUP) DUP command to make a copy of ENFORMMT (the Edit version of the default message table) and make changes to the copy. In either case, the format and contents of the Edit file must be consistent with the rules described later in this appendix under 'Required Format of the Edit File'.

2. Select a name for the key-sequenced version of the message table and use the Command Interpreter PURGE command to purge any existing file with that name.

3. Create the disc file for the key-sequenced version of the message table by issuing the following commands:

```
:FUP
-SET TYPE K
-SET EXT (32,0)
-SET REC 289
-SET IBLOCK 4096
-SET BLOCK 4096
-SET KEYOFF 0
-SET KEYLEN 34
-CREATE key-sequenced-filename
```

   where *key-sequenced-filename* is the name you have chosen for the disc file.

4. Convert the Edit file into a key-sequenced file by executing BUILDMK. To execute BUILDMK, enter:

```
BUILDMK edit-file-name, key-sequenced-filename
```

   where *edit-file-name* is the name of the disc file for the Edit version and *key-sequenced-filename* is the name of the disc file for the key-sequenced version.

Figure E-1 illustrates the preceding steps. In this example, a ?VOCABULARY section that translates the reserved words into German is created in an Edit file named *newemt*; FUP is used to create *newemk*, the key-sequenced version; and BUILDMK is used to convert *newemt* to *newemk*.

1. Create an Edit file version of the message table that contains the desired text.

```
:EDIT newemt !
*ADD 1
*    1     ?VOCABULARY
*    2     MITTELWERT=AVG, ORDNUNG=BY, ZEIGE=LIST,
*    3     UEBER=OVER, WOBEI=WHERE, MITTIG=CENTER,
*    4     ALLE=ALL, BIS=THRU, FORMAT=AS, EROEFFNE=OPEN
*  ...     ...
```

2. Purge any existing file with the same name as the name you selected for the key-sequenced version.

```
:PURGE newemk
```

3. Use the FUP to create the disc file for the key-sequenced version of the message table by issuing the following commands:

```
:FUP
-SET TYPE K
-SET EXT (32,0)
-SET REC 289
-SET IBLOCK 4096
-SET BLOCK 4096
-SET KEYOFF 0
-SET KEYLEN 34
-CREATE newemk
```

4. Execute BUILDMK to convert the Edit file version to the key-sequenced version.

```
:BUILDMK newemt, newemk
```

Figure E-1. Creating a Message Table for the Current Session

To use the message table created in Figure E-1, you must identify *newemk* on the ENFORM command:

```
:ENFORM,newemk
```

ENFORM then allows you to use the redefined words during the current session:

```
>EROEFFNE employee;
>ZEIGE ORDNUNG regnum,
       ORDNUNG branchnum,
       MITTELWERT (salary UEBER branchnum) FORMAT M<Z,ZZZ,999>,
       MITTELWERT (salary UEBER regnum) FORMAT M<Z,ZZZ,999>,
   WOBEI regnum = 1 BIS 2,
   MITTIG ALLE;
```

Guidelines For Replacing the Default Message Table

You can change the message table for all ENFORM sessions by replacing the default message table, ENFORMMK. The new message table can either contain only ?MESSAGES and ?HELP sections (ENFORM uses the new messages and help text with the standard reserved words) or ?VOCABULARY, ?MESSAGES, and ?HELP sections (ENFORM uses the new reserved words, messages and help text). Remember that changes to the default message table affect the message text for all individuals in the system who use ENFORM and do not specify their own message table. To replace the default message table, perform the following tasks:

1. Use the Command Interpreter VOLUME command to position yourself on the volume and subvolume where the default message table, ENFORMMK, resides. If you do not know the name of this volume and subvolume, ask your system manager.

2. Use the FUP DUP command to create a backup copy of ENFORMMT (the default Edit file version) and ENFORMMK (the default key-sequenced version).

3. Edit ENFORMMT and make the desired changes. Any changes to this file must conform to the rules specified later in this appendix under 'Required Format of the Edit File'.

4. Purge the current version of ENFORMMK.

5. Use FUP to create the disc file for ENFORMMK by entering the following commands:

```
:FUP
-SET TYPE K
-SET EXT (32,0)
-SET REC 289
-SET IBLOCK 4096
-SET BLOCK 4096
-SET KEYOFF 0
-SET KEYLEN 34
-CREATE ENFORMMK
```

6. Convert the Edit file version to the key-sequenced version by executing BUILDMK. To execute BUILDMK, enter:

```
:BUILDMK enformmt,emformmk
```

The example shown in Figure E-2 illustrates the preceding steps.

1. Specify the volume and subvolume on which the default message table resides in the Command Interpreter VOLUME command.

```
:VOLUME mysyst.mysyst
```

2. Make a Backup Copy of ENFORMMT and ENFORMMK.

```
:FUP DUP ENFORMMT, backupmt
:FUP DUP ENFORMMK, backupmk
```

3. Edit ENFORMMT and make the desired changes to the message table text.

```
:EDIT ENFORMMT
* ADD
     2000   ?HELP DDL
     2001   Before using ENFORM to retrieve data, you must use
     2002   DDL to describe data base fields, records, and
     2003   files. DDL creates the dictionary used by ENFORM
     2004   to obtain information about your data base.
     2005
     2006   The example below shows a DDL RECORD statement that
     ...
     2026   ?HELP Query Processor
     ...
```

4. Purge ENFORMMK.

```
:PURGE ENFORMMK
```

5. Use FUP to create the disc file for the new key-sequenced version.

```
:FUP
-SET TYPE K
-SET EXT (32,0)
-SET REC 289
-SET IBLOCK 4096
-SET BLOCK 4096
-SET KEYOFF 0
-SET KEYLEN 34
-CREATE enformmk
```

6. Execute BUILDMK.

```
:BUILDMK enformmt, emformmk
```

Figure E-2. Replacing the Default Message Table

When you run ENFORM, ENFORM uses the new version of the default message table. You need not specify the *message-table-filename* option of the ENFORM command because you have changed the system default.

Required Format of the Edit File

When you create an Edit file version of the message table, the content of the file must conform to a prescribed format. You can create an Edit file that contains:

- Only a ?VOCABULARY section. You cannot use the key-sequenced version of such an Edit file to replace the default message table; however, you can use the key-sequenced version of this file as the message table for the current ENFORM session. In this case, ENFORM uses the reserved word redefinitions from your message table and obtains message and help text from the default message table.

- Only ?MESSAGES and ?HELP sections. You can use the key-sequenced version of this file either as the message table for the current ENFORM session or to replace the default message table. In either case, ENFORM uses your message and help text with the standard reserved words, system variable names, option variable names, and command names.

- ?VOCABULARY, ?MESSAGES, and ?HELP sections. You can use the key-sequenced version of this file either as the message table for the current ENFORM session or to replace the default message table. In this case, ENFORM obtains reserved word redefinitions, message text, and help text from your message table.

The following paragraphs describe the required format and content of each section.

?VOCABULARY SECTION. The ?VOCABULARY section allows you to redefine ENFORM reserved words, system variable names, option variable names, and command names.

If you add a ?VOCABULARY section to an Edit file version of the message table, the format of this section must conform to the following rules:

1. If you include a ?VOCABULARY section, this section must be the first section in the Edit file.

2. You must enter the characters ?VOCABULARY in columns 1 through 11 of the first line of this section.

3. You can enter definition pairs on all subsequent lines of this section. Definition pairs consist of the following:

   ```
   new-element-name [ = ] old-element-name
   ```

   where *new-element-name* is the new name for the reserved word, system variable, option variable, or command and *old-element-name* is the old name.

4. When specifying definition pairs, you must enter only 7-bit ASCII characters for *new-element-name*. A *new-element-name* must conform to the naming rules specified in the *ENFORM Reference Manual.*

5. You need enter only the definition pairs for the elements being redefined. Elements that you do not redefine retain their default names.

6. You can enter more than one definition pair on a line by using a comma as a separator.

7. If you redefine the name of an option variable or a system variable, do not include the symbol @ as part of either *new-element-name* or *old-element-name.* When you specify the new name during an ENFORM session, you must include the symbol @.

8. If you redefine the name of an ENFORM command, do not include the symbol ? as part of either *new-element-name* or *old-element-name*. When you specify the new name during an ENFORM session, you must include the symbol ?.

9. If you redefine an ENFORM reserved word that applies to more than one category (statement, clause, command, aggregate, system variable, or option variable), ENFORM applies the new name to all categories. For example, if you redefine the reserved word SPACE both the SPACE clause and the @SPACE option variable are redefined.

Figure E-3 shows a diagram of the vocabulary section.

```
?VOCABULARY
new-element-name = old-element-name
...
new-element-name = old-element-name
```

Figure E-3. Diagram of the Vocabulary Section.

Refer to the *ENFORM Reference Manual* for the names of system variables, option variables, and commands. Table E-1 shows the ENFORM reserved words.

Table E-1. ENFORM Reserved Words

ACROSS *	DESC	LINK	SAVE	WHERE
AFTER	DICTIONARY	LIST	SET	WITH
ALL	ELSE	LT	SKIP	WITHOUT
AND	END	MAX	SPACE	ZERO
AS	EQ	MIN	START	ZEROS
ASCD	EQUAL	NE	SUBFOOTING	'
AT	EXIT	NOHEAD	SUBTITLE	(
AVG	FILE *	NOPRINT	SUBTOTAL	)
BEFORE	FIND	NO	SUM	*
BEGINS	FOOTING	NULL	SUPPRESS	+
BLANK	FORM	OF	TAB	–
BLANKS	GE	OFF	THAN	.
BY	GREATER	ON	THEN	/
CENTER	GT	OPEN	THRU	;
CHANGE	HELP *	OPTION	TIME	<
CLOSE	HEADING	OPTIONAL	TIMESTAMP-DATE	=
CONTAINS	IF	OR	TIMESTAMP-TIME	>
COPY	INTERNAL	OVER	TITLE	@
COUNT	INVOKE*	PARAM	TO	[
CUM	IS	PCT	TOTAL	]
DATE	JULIAN-DATE	PRINT	UNIQUE	
DECLARE	KEY	RECORD	USING *	
DEFINE *	LE	ROW-SUBTOTAL *	VIA	
DELINK	LESS	ROW-TOTAL *	WHEN *	

* These words are reserved for future extensions to ENFORM.

?MESSAGES SECTION. The ?MESSAGES section is essentially a table of text that ENFORM uses for printing error, warning, and informational messages. This message text is also used for messages that appear in the generic files QUERY-STATUS-MESSAGES and QUERY-QPSTATUS-MESSAGES.

If you either create a new ?MESSAGES section or modify the ?MESSAGES section of the Edit file version of the default message table, the ?MESSAGES section must conform to the following rules:

1. You must not include a ?MESSAGES section without including a ?HELP section.

2. In the Edit file, enter the ?MESSAGES section after the ?VOCABULARY section (if present) and before the ?HELP section.

3. You must enter the characters ?MESSAGES in columns 1 through 9 of the first line of this section.

4. For the subsequent lines of the ?MESSAGES section, enter one line of text for each message. You must enter the text in the same order and with the same number of entries as are present in the current ENFORM message text. Include the text for each message even if you do not change the message itself. If you change the order of the messages, ENFORM could print an incorrect or garbled message when an error occurs. If you omit a message, ENFORM prints ??? in place of the message text.

5. You can enter a maximum of 132 characters for the message text. The maximum length of the message depends upon where ENFORM uses the message.

Figure E-4 shows a diagram of the ?MESSAGES section.

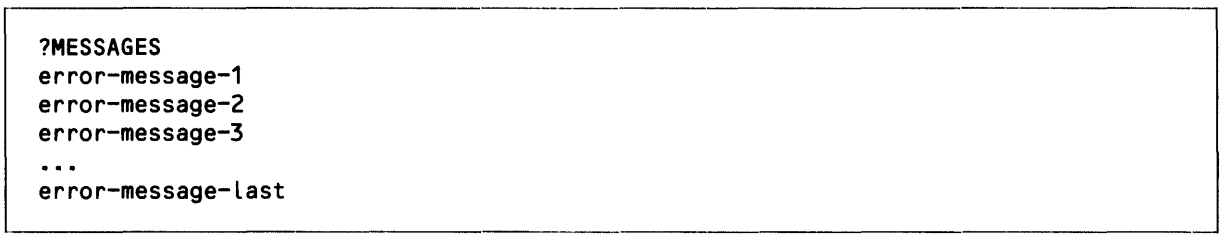


Figure E-4. Diagram of the ?MESSAGES Section

You can obtain a current copy of the ENFORM ?MESSAGES section by duplicating the ?MESSAGES section of ENFORMMT, the Edit file version of the default message table. An asterisk that appears within the ?MESSAGES section of ENFORMMT represents a table entry reserved for future messages. To preserve the order of the message text, include any asterisks appearing in ENFORMMT in your new ?MESSAGES section.

?HELP SECTION. The ?HELP section contains the text that ENFORM displays when the ?HELP command is issued. This required section is divided into several subsections with one subsection for each of the items for which help is available. Each subsection of the ?HELP section contains the lines which ENFORM displays when a specific form of the ?HELP command is entered.

If you either create a new ?HELP section or modify the ?HELP section in the Edit file version of the default message table, the ?HELP section must conform to the following rules:

1. You must not include a ?HELP section without including a ?MESSAGES section.

2. In the Edit file, enter the ?HELP section after both the ?VOCABULARY section (if present) and the ?MESSAGES section.

3. You must enter the characters ?HELP in columns 1 through 5 of the this section (which is also the first line of the first ?HELP subsection).

4. Enter the list of all elements for which help is available in subsequent lines of the first subsection. ENFORM displays this list when ?HELP is entered.

5. If you add or delete subsequent subsections, add or delete the associated element from the list of elements in the first subsection.

6. Enter the ?HELP *help-element* in the first column of the first line of all subsequent subsections, where help-element is the name of the element for which help is available. When you enter *help-element*, you must follow the naming rules described in the *ENFORM Reference Manual* with one exception: you can use a question mark (?) as the initial character when *help-element* is a command. The length of *help-element* cannot exceed 31 characters.

7. Enter the help text for *help-element* in the subsequent lines of the subsection. ENFORM displays this text when you enter ?HELP *help-element*.

8. ENFORM imposes no limit on the length of help text; however, avoid specifying help text that is longer than 23 lines. If you specify text that is longer than 23 lines, reading the text will be extremely difficult for individuals who request help from terminals that have no scrolling mechanism (for example, the Tandem 6510 terminal).

9. ENFORM imposes no restrictions on the content of help text. You can add a ?HELP subsection for any topic.

Figure E-5 shows a diagram of the ?HELP section.

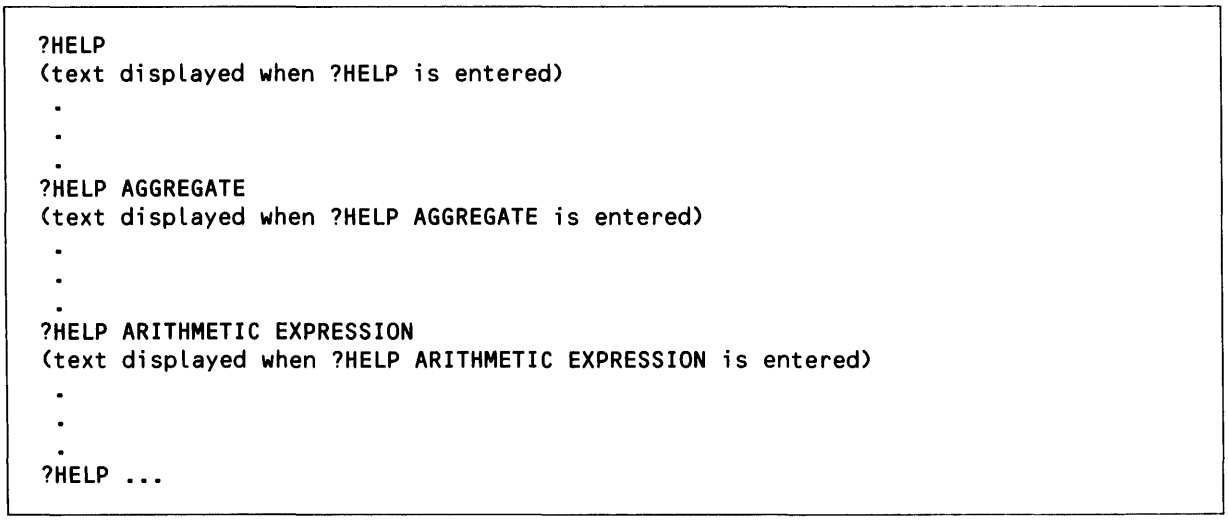


Figure E-5. Diagram of the ?HELP Section

You can obtain a copy of the current help text by duplicating ENFORMMT, the Edit file version of the default message table.

APPENDIX F

GLOSSARY

Aggregate—a cumulative operation on set(s) of numbers, producing a single value per set. See Predefined Aggregate and User Aggregate.

By-item—the field name used to group and sort ENFORM output; always associated with a BY or BY DESC clause. A by-item is a special kind of target-item.

Clause—component of an ENFORM statement.

Command—a directive to the ENFORM compiler.

Compiler/Report Writer—the ENFORM process that both compiles ENFORM queries and formats and writes ENFORM reports.

Compiled Query File—the physical file containing a saved query that been compiled by the ?COMPILE command.

Composite Key—two or more contiguous fields that can be used to identify a record occurrence.

Current Output Listing File—the file to which ENFORM directs output; this file can change during an ENFORM session.

Data Base—a set of related files defined in a dictionary.

Data Category—the type of data that can be stored in a field; either alphanumeric or numeric.

Data Definition Language (DDL)—the language used to describe the record and file structure of a data base.

Dictionary—a data base of file descriptions and record types created by the Data Definition Language (DDL); also called data dictionary.

Default Output File—the file to which ENFORM directs output at the beginning of an ENFORM session. See also Current Output Listing File.

Default Input File—the file from which the ENFORM source code is entered when the IN option of the ENFORM command is omitted; usually the home terminal.

Elementary Field—smallest named unit of a record.

ENFORM Server—a user written process that can supply data to a query processor as an alternative to the data being supplied directly from a disc file.

Field—either an elementary field or group field.

Field Name—name given to a field in a DDL RECORD statement.

Field Value—value of a specific field within a specific stored record.

File—a collection of similarly structured records.

File Name—the name of a physical file.

File Type—identifies the organization of the physical file, such as key-sequenced, entry-sequenced, relative, or unstructured.

Front End—the ENFORM process which compiles ENFORM programs, and prints reports. See Compiler/Report Writer.

Generic File—a file used to store some class of ENFORM output.

Group Field—a collection of one or more fields that can be accessed with a single name.

Group Name—name of one or more fields that can be accessed with a single name.

Home Terminal—the terminal from which the ENFORM command is entered.

Link—specifies a relationship between records in a relational data base to be used in an ENFORM query.

Literal—one or more numeric or alphanumeric characters. See String Literal and Numeric Literal.

Logical Expression—an expression that returns a true or false value.

Normalized—data that has been described in such a manner that only one value exists for every field position in a record.

Numeric Literal—composed of the digits 0 through 9. Numeric literals cannot be larger than 32765 and must be enclosed in parentheses unless they appear in a logical expression or a TAB, SPACE, SKIP, or FORM clause.

Option Variable—An ENFORM-supplied variable that defines certain operational values.

OUT File—the physical device specified in the OUT option of the ENFORM command.

Physical File Name—GUARDIAN file name in the form *\system-name.volume-name.subvol-name.file-name.*

Predefined Aggregate—one of the ENFORM aggregates: AVG, COUNT, MAX, MIN, or SUM.

Primary Key—the field or group of fields that uniquely identifies a record.

Program—a sequence of related ENFORM commands and statements.

Qualification Aggregate—an aggregate that appears in a request-qualification. See also Aggregate.

Qualified Field Name—a name that uniquely identifies a field as a component of a record description.

Query—a complete ENFORM LIST or FIND statement specifying which fields and records to retrieve.

Query Processor (QP)—the ENFORM process that opens the files and retrieves the records from a relational data base for a report or a new file.

Record—a related set of field values.

Record Description—description of a record in a data base, containing the record name, the file name and type, and key definitions.

Record Name—name given a record description in a DDL RECORD statement.

Record Occurrence—the actual stored values associated with a record.

Record Type—a record's structure including field names and data types.

Relational Data Base—a data base in which records are related through fields with common formats and comparable values.

Repeating Group—a data base field that contains more than one data value.

Report—the printed output of an ENFORM query using an ENFORM LIST statement.

Request-qualification—the condition or conditions that a data base element must satisfy to contribute to the target-record; begins with a WHERE clause followed by a logical expression.

Reserved words—keywords with specific meaning and reserved by ENFORM.

Server Query Processor—specific query processor identified by an ?ATTACH command, initiated separately from the dedicated query processor started by the ENFORM compiler.

Session—period of interaction with ENFORM.

Source Code—the ENFORM statements, clauses, and commands that comprise the query specifications.

Source File—the Edit file that contains the source code. See also Source Code.

Statement—main instruction of an ENFORM program.

String Literal—one or more alphanumeric characters enclosed in quotation marks (" ").

Subscript—a value used to select a particular element.

System Variable—an ENFORM-supplied variable that returns the current time, date, line number, and page number.

Target Aggregate—an aggregate that appears as a part of the target-record.

Target-file—the file produced by the Query Processor that contains records with all the information requested in the query specifications.

Target-item—the record names, field names, expressions, variables, aggregates, and literals, including by-items, whose values appear in a target-record.

Target-list—the record names, field names, expressions, variables, aggregates, and literals following the keywords LIST or FIND that contribute to the target-record. Target-lists consist of target-items some of which are by-items. See also By-items and Target-items.

Target-record—the records generated by the Query Processor from which your ENFORM output is produced.

Unnormalized—data that has been described such that more than one value exists for a field position in a record.

User Aggregate—a user-declared aggregate; a user-defined function that returns a value. See Aggregate.

User Variable—a user-declared element that can be used to store numeric or string literals, field values, or the results of arithmetic or aggregate calculations.

?OUT File—the physical device specified in the ?OUT command.

INDEX

YOUR COMMENTS PLEASE

Tandem NonStop™ & NonStop II™ Systems
ENFORM™ User's Guide
82195 (Update 1 to 82349 B00)

Tandem welcomes your comments on the quality and usefulness of its publications. Does this publication serve your needs? If not, how could we improve it? If you have specific comments, please give the page numbers with your suggestions.

This comment sheet is not intended as an order form. Please order Tandem publications from your local Sales office.

FROM:

Name _____ Date _____

Company _____

Address _____

City/State _____ Zip _____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 482 CUPERTINO, CA, U.S.A.

POSTAGE WILL BE PAID BY ADDRESSEE

Tandem Computers Incorporated
19333 Vallco Parkway
Cupertino, CA 95014-9990

Attn: Manager—Software Publications

TAPE

TAPE