Tandem NonStop™ & NonStop II™ Systems

ENFORM™ Reference Manual

ABSTRACT: This manual provides detailed information about the syntax of the ENFORM language.

PRODUCT VERSION: ENFORM C11

OPERATING SYSTEM VERSION: GUARDIAN A05 (NonStop II System)
GUARDIAN E06 (NonStop System)

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

Part No. 82348 B00

April 1983
Printed in U.S.A.

October 1982 Manual released.

April 1983 Revised.

NEW AND CHANGED INFORMATION

This revision documents the HELP command and changes to the FIND statement. Miscellaneous technical and editorial corrections are also included.

CONTENTS

FIGURES

TABLES

PREFACE

This manual is one of three volumes that describe the ENFORM language. This manual, which concerns syntax only, should be used as a reference by experienced ENFORM users. For other information about ENFORM and related products, refer to the publications listed in alphabetical order below.

Data Definition Language (DDL) Programming Manual

EDIT Manual

ENFORM User's Guide

ENSCRIBE Programming Manual

GUARDIAN Operating System Command Language and Utilities Manual

GUARDIAN Operating System Programming Manual—Volumes 1 and 2

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

This publication documents:

- The syntax of the Command Interpreter ENFORM command needed to call the ENFORM process, the Command Interpreter commands needed to create a server query processor, and the names of generic files that can be assigned using either the Command Interpreter ASSIGN command or the ENFORM ?ASSIGN command.
- The syntax of the ENFORM language elements including statements, clauses, commands, aggregates, expressions, literals, and variables. These elements are alphabetized wherever appropriate.
- The error messages generated by ENFORM.

The special features of ENFORM, such as using the host language interface, writing an ENFORM server, or redefining ENFORM reserved words and message text are not discussed. For this information and for information about using the ENFORM language elements, refer to *Introduction to ENFORM* or to the *ENFORM Users Guide*.

The examples in this manual use the data base described in the *ENFORM Users Guide*.

ENFORM TERMINOLOGY

This manual uses special terms to describe the various components and features of the ENFORM language. Figure 1-1 illustrates these terms by showing a typical ENFORM session (the period of time that begins when you enter the ENFORM command and ends when you exit ENFORM). The following paragraphs describe some of the terms illustrated in Figure 1-1. Become familiar with these terms, because they are used throughout the manual.

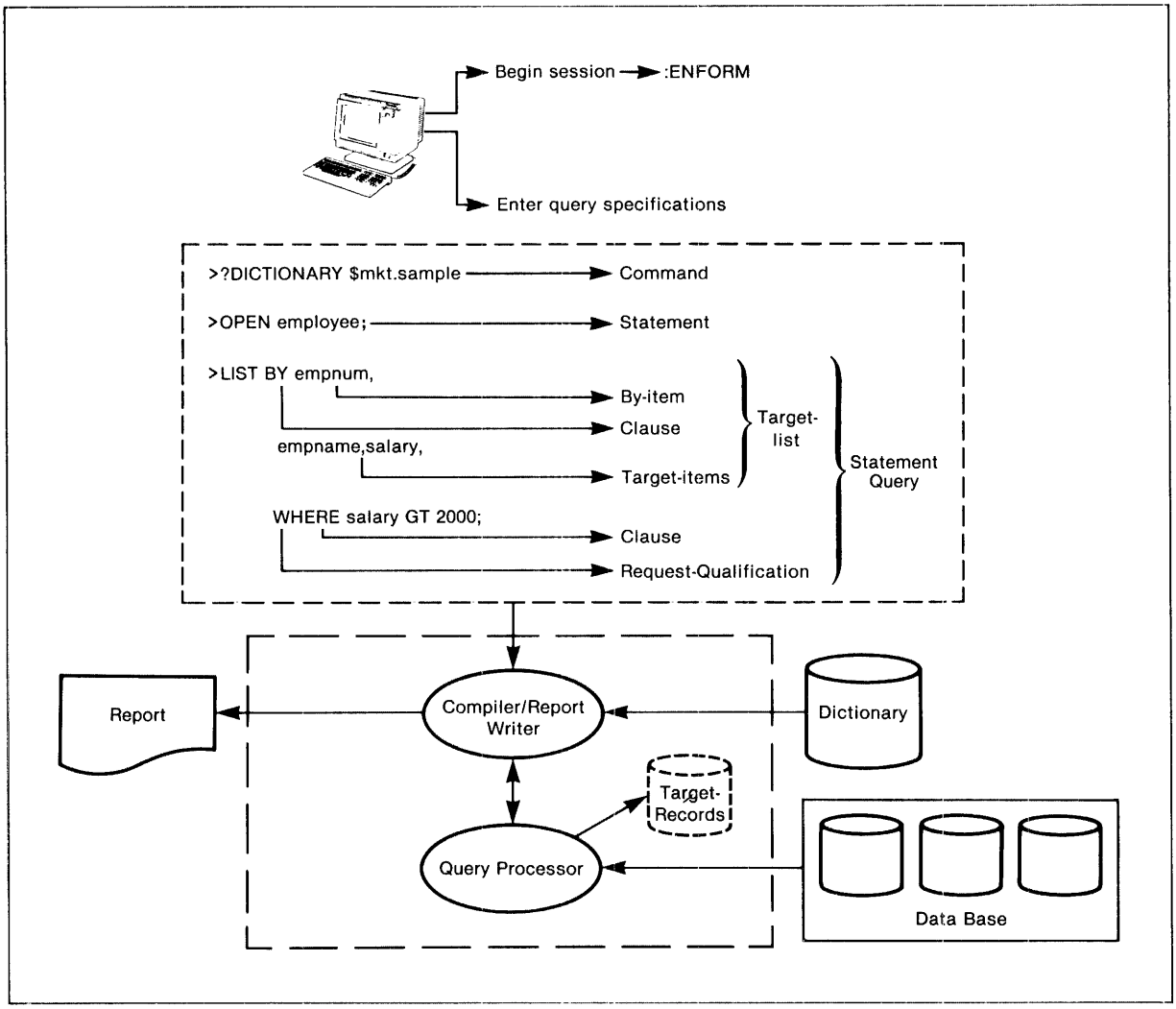


Figure 1-1. Typical ENFORM Session

The important terms are:

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 statements specify the information to be retrieved.

Target-list—a part of the query; a target-list consists of target-items and a special type of target-item called a by-item.

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

By-items—field names modified by the BY or BY DESC clauses described in Section 5. 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 keyword followed by a logical expression.

Compiler/report writer—the ENFORM process that compiles your query and writes a report (if one is requested). The compiler/report writer issues error messages for syntax errors. If no errors exist, the compiler/report writer passes your compiled query specifications along with information obtained from the dictionary to the query processor. After the query processor returns the retrieved data (in the form of the target-records described later in this section), the compiler/report writer formats and writes the report.

Dictionary—physical files that contain information called record descriptions. A record description provides ENFORM with information about the name of the record being accessed, the name and data type of any fields within the record, the record and field length, the name of any primary or alternate key fields, and the name of the physical file associated with the record description. The dictionary is created by the Data Definition Language (DDL) compiler from record descriptions written in DDL. The dictionary must exist before your query specifications are processed. Refer to the *Data Definition Language (DDL) Reference Manual* and the *ENFORM User's Guide* for more information about the dictionary.

Query processor—the ENFORM process that uses the information provided by the compiler/report writer to retrieve information from the data base. The query processor also performs other functions such as creating a new physical file and transmitting records to a host language program. Creating a new physical file is described with the FIND statement in Section 5 of this manual. The host language interface is described in the *ENFORM User's Guide*.

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

Data base—the collection of physical files from which the query processor retrieves data. Any physical file from which data is retrieved must be associated with the record description obtained from the dictionary.

SECTION 2

RUNNING ENFORM

The ENFORM command issued from the Command Interpreter calls the ENFORM subsystem. If no parameters are specified, the ENFORM command appears as:

```
:ENFORM
```

The terminal from which the ENFORM command is entered is called the home terminal. The syntax for the ENFORM command is:

```
ENFORM [ / [ IN input-filename ] [ , [  OUT output-filename ] ] / ]
  [ dict-subvol-name ] [ , message-table-filename ]
```

where

`IN input-filename`

specifies either the name of an EDIT file containing ENFORM source code or the name of a compiled query file.

If this option is specified, ENFORM executes the code in the specified file and returns you to the Command Interpreter prompt at the end of execution.

If this option is omitted, the ENFORM prompt (>) appears and you can enter commands and statements either directly by typing them or indirectly by specifying either the ?RUN or ?SOURCE commands. When this option is omitted, the home terminal becomes the default input file.

`OUT output-filename`

specifies the name of the physical file to which output is directed. If this option is omitted, ENFORM directs the output to the current output listing file (explained later in this section.)

`dict-subvol-name`

is the name of the volume and subvolume upon which the dictionary resides. If this parameter is omitted, ENFORM assumes the dictionary resides on the current volume and subvolume. Either the DICTIONARY statement or the ?DICTIONARY command can be used to supply or change the dictionary name. →

message-table-filename

is the name of the key-sequenced file containing a user-defined ENFORM message table. ENFORM retrieves error and informational message text, help message text, and/or the list of any redefined reserved words, system variables, option variables, or command names from this file when this parameter is specified. If this parameter is omitted, ENFORM retrieves message text from a message table supplied by Tandem. Refer to the *ENFORM Users Guide* for information about creating a user-defined message table.

The Command Interpreter ASSIGN command (described later in this section) can be used to make up to 32 logical file assignments before the ENFORM subsystem is called.

INTERACTIVE MODE

ENFORM functions in interactive mode when the ENFORM source code is entered from a terminal keyboard. ENFORM prompts for input by printing the right angle bracket (>). When a carriage return is entered, ENFORM issues another prompt. For example:

```
:ENFORM
ENFORM  - T9102C09 - (02APR82)  DATE - TIME: 6/16/81 - 10:14:52
>
```

ENFORM commands and statements can be entered either directly or indirectly. Commands and statements are entered directly when you type them in response to the ENFORM prompt. Commands and statements are entered indirectly when you use either the ?RUN or ?SOURCE commands to supply the ENFORM source code.

When you enter commands and statements directly, the Command Interpreter FC command provides you with the ability to edit or repeat a line. Refer to the *GUARDIAN Operating System Command Language and Utilities Manual*, for more information about this command.

When you specify the ?EDIT command, the Edit prompt (*) appears and all the functions of the Edit process are available for your use. Either the ?RUN or the ?SOURCE command can be used to execute the source code created in the Edit process. These commands are described in Section 6.

Exit interactive mode by entering the EXIT statement, the ?EXIT command, or by pressing the CTRL and Y terminal keys simultaneously.

NONINTERACTIVE MODE

ENFORM functions in noninteractive mode when commands and statements are entered through an input file other than a terminal. The input file can be an edit file or a compiled query file. For example:

```
:ENFORM /IN input-filename, OUT output-filename/
```

In this example, ENFORM reads the commands and statements from the input file.

When ENFORM functions in noninteractive mode, all commands, statements, and clauses must be part of the input file. When the file specified in the IN option is a compiled query file, values for parameters can be specified by using the PARAM command of the Command Interpreter prior to the ENFORM command. More information about passing parameters to compiled ENFORM queries is available later in this section.

ENFORM terminates when an end-of-file, EXIT statement, or ?EXIT command is encountered on the input file.

THE CURRENT OUTPUT LISTING FILE

The current output listing file is the file to which ENFORM directs output. During the course of an ENFORM session, the current output listing file can change.

At the beginning of an ENFORM session, the current output listing file is the default output file. ENFORM determines the default output file by the following process:

- If the OUT option is included in the ENFORM command, the default output file is the file specified in the OUT option.
- If the OUT option is omitted from the ENFORM command, the default output file is the file specified in the IN option of the ENFORM command if that file is a terminal.
- If the file specified for the IN option is not a terminal, the default output file is the home terminal.
- If both the IN option and the OUT option are omitted from the ENFORM command, the default output file is the home terminal.

As the session progresses, the current output listing file might change as follows:

- If a QUERY-COMPILER-LISTING file is specified, that file becomes the current output listing file whenever ENFORM commands and statements are being processed.
- If either a QUERY-REPORT-LISTING file or an ?OUT command file is specified, that file becomes the current output listing file whenever a report (the output from a LIST statement) is being processed.

The QUERY-COMPILER-LISTING file and the QUERY-REPORT-LISTING file are discussed later in this section. The ?OUT command file is discussed in Section 6.

PRESSING THE TERMINAL BREAK KEY

ENFORM acknowledges the terminal BREAK key whenever the input file is a terminal. The input file is a terminal under the following conditions:

- the IN option is omitted from the ENFORM command making the home terminal the default input file.
- the terminal (whether the home terminal or another terminal) is the file specified in the IN option of the ENFORM command.

Pressing the terminal BREAK key on any terminal, including the home terminal, has no effect unless the terminal is the input file.

The current activity determines the action taken as follows:

- Pressing the terminal BREAK key when ENFORM is processing a query, producing a report, or producing output from the ?SHOW command, returns the terminal to the ENFORM prompt (>). Query execution and any output (the report or the output produced by the ?SHOW command) terminates.
- Pressing the terminal BREAK key when commands or statements are being entered (either directly or indirectly), returns the terminal to the Command Interpreter prompt (:).

- Pressing the terminal BREAK key when you have entered the EDIT process from the ENFORM prompt has the same effect as when the EDIT process is entered from the Command Interpreter prompt with one exception: pressing the terminal BREAK key at the EDIT prompt (*) has no effect.

If the @BREAK-KEY option variable (described in Section 5) is set to OFF, pressing the BREAK key returns the terminal to the Command Interpreter prompt. It neither terminates output nor query execution. In this case, ENFORM continues to run and any output directed to the terminal is temporarily interrupted. The PAUSE command of the Command Interpreter can be issued to resume output.

LOGICAL FILE ASSIGNMENTS

ENFORM allows logical file assignments to be made either before or after the Command Interpreter ENFORM command is entered. When the logical file assignment is made from within the ENFORM subsystem, use the ENFORM ?ASSIGN command described in Section 6. When the logical file assignment is made before the ENFORM command is entered, use the Command Interpreter ASSIGN command to either associate a physical file with a dictionary record description or to assign some form of ENFORM output to a generic file.

When ENFORM is used in noninteractive mode, the Command Interpreter ASSIGN command overrides an ENFORM ?ASSIGN command that is part of the input file. When ENFORM is used in interactive mode, the ENFORM ?ASSIGN command overrides any Command Interpreter ASSIGN commands.

The syntax of the Command Interpreter ASSIGN command is:

```
ASSIGN[ {record-name      } , physical-filename [ , exclusion-mode ] ]
       [ {generic-file-name}                                       ]
```

where

`record-name`

is the name of a dictionary record description.

`generic-file-name`

is the name of one of the following generic files: QUERY-WORK-AREA, QUERY-SORT-AREA, QUERY-QPSTATISTICS, and QUERY-QPSTATUS-MESSAGES. All other names are ignored.

`physical-filename`

is a fully qualified Tandem file name. Refer to the *GUARDIAN Operating System Programming Manual* for the exact form of a Tandem file name.

`exclusion-mode`

is either SHARED, PROTECTED, or EXCLUSIVE. The default is SHARED for an input file. Valid values for generic files are described later in this section.

Refer to the *GUARDIAN Operating System Programming Manual* for more information about the ASSIGN command.

PASSING PARAMETERS TO COMPILED QUERY FILES

ENFORM allows you to pass parameters to a compiled query file. The compiled query file must contain a PARAM statement defining the parameter. The PARAM statement is discussed in Section 4.

Use the PARAM command of the Command Interpreter to specify parameters prior to execution of the compiled query file. The PARAM command overrides any values specified in a SET statement for a parameter. The syntax of the Command Interpreter PARAM command is:

```
PARAM [ parameter-name  parameter-value ] ,...
```

where

parameter-name

is the name of a parameter defined in a PARAM statement.

parameter-value

is the value to be assigned to *parameter-name*. *parameter-value* can have either of the following forms:

```
character-string
"character-string"
```

If the first form is used, the string must not contain any embedded commas, and leading and trailing blanks are not included as part of *parameter-value*.

If the second form is used, all the characters, including leading and trailing blanks, between the quotation marks are included as part of *parameter-value*.

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

A SERVER QUERY PROCESSOR

Unless otherwise specified, each ENFORM session uses a dedicated query processor. To avoid some overhead, several compiler/report writer processes can be assigned to share a single server query processor and sort process. The assignment is made by an ?ATTACH command. A server query processor processes one query at a time. Figure 2-1 shows several compiler/report writer processes assigned to one server query processor.

One or more server query processors can be created. Each server query processor runs NonStop, handling queries from one compiler/report writer process at a time.

A host language program (described in the *ENFORM Users Guide*) can also use a server query processor.

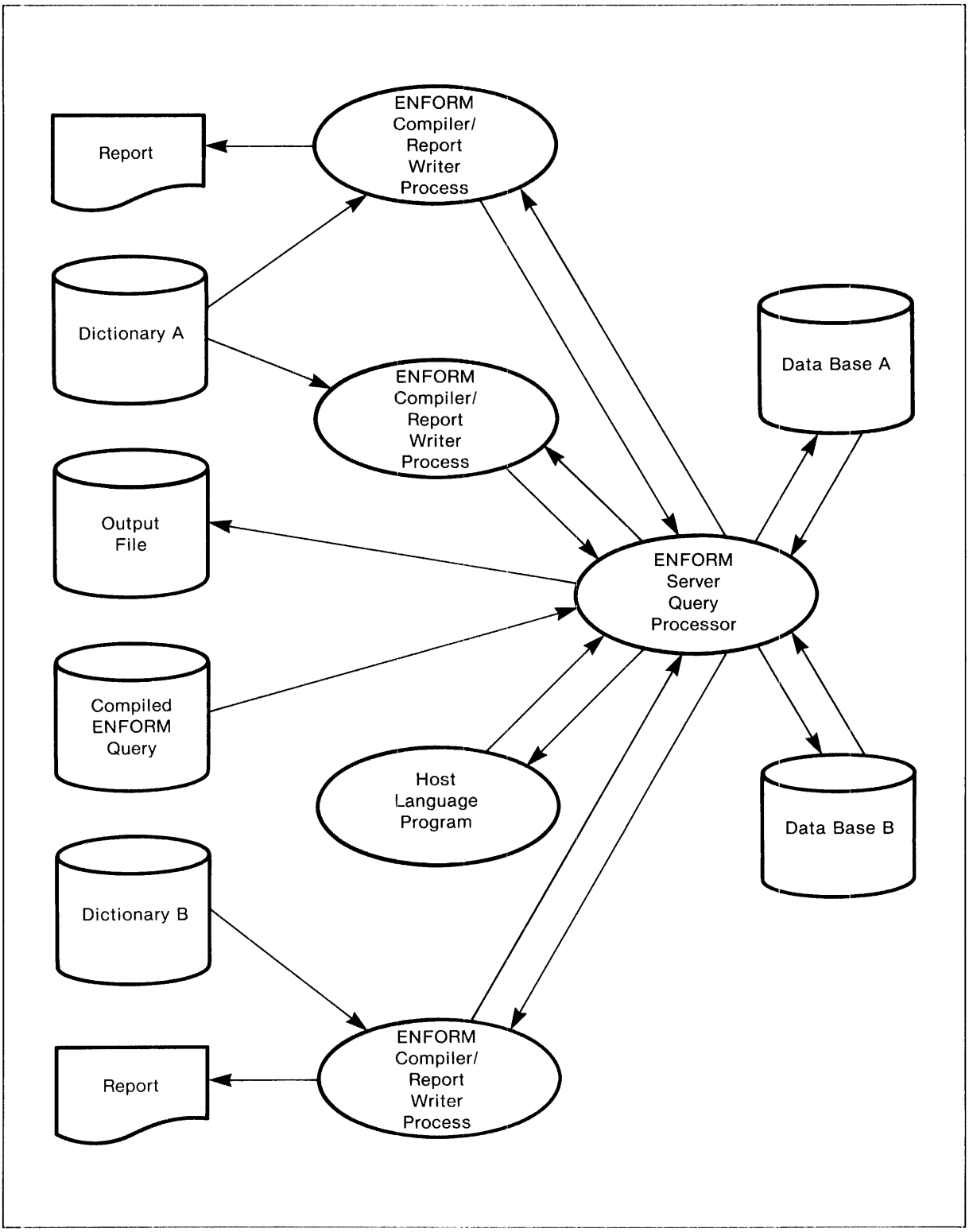


Figure 2-1. Server Query Processor With Several Compiler/Report Writer Processes

ENFORM server query processors are created by a system manager, using the Command Interpreter ASSIGN and PARAM commands followed by the Command Interpreter QP command.

The Command Interpreter ASSIGN Command

For every physical file an ENFORM application accesses, the server query processor must do open/close operations that add to the processing time. This processing time can be reduced significantly if ENFORM applications that frequently use the same physical files are processed by a common server query processor. A server query processor allows for heavily accessed physical files to be kept open.

The ASSIGN command of the Command Interpreter defines the physical file or files kept open. The syntax is:

```
ASSIGN { Fnumber          } , physical-filename [ , exclusion-mode ]
       { generic-file-name}
```

where

Fnumber

specifies the logical file name. Legal values for logical file names range from F1 through F31.

generic-file-name

is the name of one of the following generic files: QUERY-WORK-AREA, QUERY-SORT-AREA, QUERY-QPSTATISTICS, and QUERY-QPSTATUS-MESSAGES. All other names are ignored.

physical-filename

is a fully qualified Tandem file name. Refer to the *GUARDIAN Operating System Programming Manual* for the exact form of a Tandem file name.

exclusion-mode

is either SHARED, PROTECTED, or EXCLUSIVE. The default is SHARED for an input file. Valid values for generic files are described later in this section.

The Command Interpreter PARAM Command

To create a server query processor, you must include a PARAM command specifying a REQUESTORS parameter. The REQUESTORS parameter defines the maximum number of requestors the server query processor can accept. The other parameters are optional and can be specified in the same or in a different PARAM statement.

```
PARAM REQUESTORS max-requestors [,COST max-cost     ]
                                [,TIMEOUT time-out  ]
                                [,READS max-reads   ] ,...
                                [,CPU number        ]
```

where

`REQUESTORS max-requestors`

sets the maximum number of requestors a query processor can accept. A requestor can be any ENFORM application or a host language program. (Host language programs are described in the *ENFORM Users Guide).*

`COST max-cost`

sets a strategy cost limit for each ENFORM query using this server query processor. If an ENFORM query exceeds the limit, it is terminated and an error message is displayed. Refer to the @COST-TOLERANCE option variable in the Option Variable clauses in Section 5 for an explanation of strategy cost limits.

Max-cost must be an integer between one and eight. The default is no limit.

`TIMEOUT time-out`

sets the number of minutes a server query processor sits idle before stopping itself. The default is no limit, meaning the query processor continues to run indefinitely.

`READS max-reads`

sets the maximum number of logical data base reads per ENFORM session. If that many reads are performed, the ENFORM session is terminated and an error message is displayed.

Max-reads must be an integer. The default is no limit.

`CPU number`

is an integer value that sets the number of the CPU where a server query processor resides.

The Command Interpreter QP Command

After the parameters are initialized and the physical files to be kept open are specified, the server query processor is created. The syntax is:

```
QP / NOWAIT, NAME process-name [ , CPU number ]
   [ , PRI priority ]  [ , MEM pages ]  /
```

where:

NAME process-name

is a process name for the server query processor. The name must begin with a dollar sign ($) followed by an alphabetic character and one to four alphanumeric characters.

CPU number

is the number of the CPU where this server query processor resides. The default is the same CPU where the Command Interpreter resides. A *CPU number* specified in the QP command overrides a CPU number specified in the PARAM command.

PRI priority

is the priority at which this server query processor is to run.

MEM pages

is the maximum number of virtual data pages used for this server query processor. It must be an integer from one to 64. The default is 64.

Example of Server Query Processor Creation

Issue the following instructions to create a server query processor named *$qp1* that keeps open *parts*, *order*, and *odetail*, accepts up to 15 requestors, sets a limit of 2 on the cost strategy, and waits idle up to 3 minutes before stopping.

```
:ASSIGN F1, $data.database.parts
:ASSIGN F2, $data.database.order
:ASSIGN F3, $data.database.odetail
:PARAM REQUESTORS 15, COST 2, TIMEOUT 3
:QP / NOWAIT, NAME $qp1 /
```

GENERIC FILES

A generic file is a file that is used to store some form of ENFORM output. ENFORM produces many different forms of output, such as statistics and error messages. To enable control over each class of output, generic file names have been defined for each class. These generic file names can be used in Command Interpreter ASSIGN commands or ENFORM ?ASSIGN commands (described in Section 6) as the record-name to be assigned to a physical-file-name.

When you assign an ENFORM generic file to a physical file, the physical file must exist at the time ENFORM attempts to open the file. If you specify an exclusion mode, it is used. An unspecified or meaningless exclusion mode (for example, protected for terminals) causes the default (SHARED for terminals and EXCLUSIVE for other devices) to be used. If you specify an exclusion mode for the generic files QUERY-WORK-AREA and QUERY-SORT-AREA, it is ignored.

Assigning a generic output file name to a process name causes the process to be treated as if it were the spooler. ENFORM OPENs the process, calls SETMODE, and WRITEs to the process. It is not possible to ASSIGN a FIND file to a process because ENFORM will create an unstructured file and rename the file as the last step in processing the FIND statement. Table 2-1 shows the ENFORM generic file names and their uses.

Table 2-1. ENFORM Generic Files and Their Uses

Generic File Name	Use
QUERY-COMPILER-LISTING	All compilation output produced during an ENFORM session (that is, entering a query either directly or indirectly); compiler errors and warnings; including output, errors, and warnings from other ENFORM commands (such as ?ASSIGN and ?SHOW). Output is produced in ASCII with a record length of 132 bytes.
QUERY-REPORT-LISTING	All reports produced as a result of the execution of a LIST statement. Output is produced in ASCII with a record length of 132 bytes.
QUERY-STATISTICS	All statistics produced during an ENFORM session by the query processor while processing a FIND or LIST statement when @STATS is SET to ON. Several sets of statistics can be produced during a session. A set is identified by the requestor's PID and the beginning and ending times of the query execution. Output is produced in ASCII. For each query processed, the output consists of (72 * (the number of record-types in the query + 3)). For display purposes, each output record is 72 characters long.
QUERY-STATUS-MESSAGES	All compiler/report writer error and warning messages produced during an ENFORM session. All error messages produced during an ENFORM session by the query processor or SORT while processing a FIND or LIST statement. These messages also appear in the listings. The output is produced in ASCII with a record length of 132 bytes.

Table 2-1. ENFORM Generic Files and Their Uses (Concluded)

Generic File Name	Use
QUERY-WORK-AREA	The volume where all temporary files (except SORT work files) are built by the query processor while processing a FIND or LIST statement during an ENFORM session. The QUERY-WORK-AREA file name should contain only a volume name.
QUERY-SORT-AREA	The location where all temporary files are built by the SORT process while processing a FIND or LIST statement during an ENFORM session. The QUERY-SORT-AREA file name can be a volume name or an explicit file name.
QUERY-QPSTATISTICS	The statistics produced for every FIND or LIST statement that is successfully processed by the query processor during an ENFORM session (regardless of the setting of @STATS). Each set of statistics is identified by a line containing the requestor's PID and the beginning and ending times of the query execution. If @STATS is set to ON, statistics are also reported to the QUERY-STATISTICS file. The output is produced in ASCII. For each query processed, the output consists of (72 * (the number of record-types in the query + 3)). For display purposes, each output record is 72 characters in length.
QUERY-QPSTATUS-MESSAGES	All error messages produced by the query processor or SORT during the processing of a FIND or LIST statement during an ENFORM session. To identify the query in error, each message is preceded by the requestor's PID and the time of the error. Errors are also reported to the QUERY-STATUS-MESSAGES file. The output is produced in ASCII with a record length of 132 bytes.

In most cases, a generic output file is an unstructured file. When ENFORM opens an unstructured file to write records, the records are written starting at the beginning of the file and the existing records are overwritten.

Generic Files and a Dedicated Query Processor

The generic files QUERY-WORK-AREA, QUERY-SORT-AREA, QUERY-QPSTATISTICS, and QUERY-QPSTATUS-MESSAGES can be specified for a dedicated query processor by using either the ENFORM ?ASSIGN command (see Section 6) or the Command Interpreter ASSIGN command (see Logical File Assignments earlier in this section). The ASSIGN command is passed through the ENFORM compiler/report writer to the query processor for each LIST or FIND statement until the ASSIGN is cleared.

The query processor opens both QUERY-QPSTATISTICS and QUERY-QPSTATUS-MESSAGES for every FIND or LIST request. Generally, statistics and errors for multiple requests to a dedicated query processor cannot be collected in a disc file because the results are written over each other. However, by assigning the generic file to a process, the statistics and errors can be collected for multiple requests to a dedicated query processor.

Generic Files and the Server Query Processor

The generic files QUERY-WORK-AREA, QUERY-SORT-AREA, QUERY-QPSTATISTICS, and QUERY-QPSTATUS-MESSAGES can be held open for the lifetime of a server query processor if the Command Interpreter ASSIGN command (see A Server Query Processor earlier in this section) is specified at the time the server query processor is created.

By using the generic file names with a server query processor, you can specify where the server query processor: builds the temporary files, performs sorts, sends statistics, and sends error messages. Since the QUERY-QPSTATISTICS and QUERY-QPSTATUS-MESSAGES files are held open for the lifetime of the associated server query processor, collection of statistics and errors for multiple requests is possible.

The Command Interpreter ASSIGN commands specified when the server query processor is started cannot be overridden by either ENFORM ?ASSIGN commands nor Command Interpreter ASSIGN commands specified when the ENFORM process is initiated. (The server query processor must be stopped and restarted to alter these file assignments.)

If you assign generic files by specifying either the ENFORM ?ASSIGN command (during ENFORM processing) or the Command Interpreter ASSIGN command (before ENFORM is initiated) the server query processor directs output to these generic files until you clear the ASSIGN command. Generic files that were not assigned when the server query processor was created are re-opened for each FIND or LIST statement.

Generic Files and the Current Output Listing File

If the Generic files QUERY-REPORT-LISTING and QUERY-COMPILER-LISTING are assigned, these files become the current output listing files under the following conditions:

- QUERY-COMPILER-LISTING file is the current output listing file whenever ENFORM statements and commands are processed.
- QUERY-REPORT-LISTING file is the current output listing file whenever a report (the output from the LIST statement) is produced unless the ?OUT command specifies an ?OUT file.

Assignment of any of the other generic files does not affect the current output listing file.

Table 2-2, shows the forms of ENFORM output (that is, reports, statement and command output, statistics, and error messages) and the corresponding output file when generic files are assigned. The default output file is described earlier in this section.

Table 2-2. ENFORM Output Files

ENFORM Output	Output File
ENFORM banner and trailer Commands Statements Command output (e.g.,?SHOW)	1) The QUERY-COMPILER-LISTING file if assigned, otherwise 2) to the default output file
Report from a LIST statement	1) The ?OUT file if specified, otherwise 2) QUERY-REPORT-LISTING file if assigned, otherwise 3) to the default output file
Statistics @STATS on	The QUERY-QPSTATISTICS file if assigned and, in addition, to the following: 1) QUERY-STATISTICS file if assigned, otherwise 2) QUERY-COMPILER-LISTING file if assigned, otherwise 3) to the default output file.
@STATS off	1) The QUERY-QPSTATISTICS file if assigned, otherwise 2) the output is not written.
Error messages from the command processor and query compiler	The QUERY-STATUS-MESSAGES file if assigned and, in addition, to the following: 1) QUERY-COMPILER-LISTING if assigned, otherwise 2) to the default output file. Note that if the QUERY-STATUS-MESSAGES file is not assigned and the QUERY COMPILER-LISTING file is not the default output file, the error messages are also written to the default output file.
Error messages from the report writer	The QUERY-STATUS-MESSAGES file if assigned, and, in addition, to the following: 1) ?OUT file if specified, otherwise 2) QUERY-REPORT-LISTING if assigned, otherwise 3) to the default output file. Note that if the QUERY-STATUS-MESSAGES file is not assigned and the list file is not a the default output file, the error messages are also written to the default output file.
Error messages from the Query processor or SORT	The QUERY-QPSTATUS-MESSAGES file if assigned, and to the QUERY-STATUS-MESSAGES file if assigned, and in addition to the following: 1) QUERY-COMPILER-LISTING if assigned, otherwise 2) to the default output file. Note that if the QUERY-STATUS-MESSAGES file is not assigned and the current output listing file is not the default output file, the error messages are also written to the default output file.

The numbers in the Output File column indicate the order in which ENFORM directs output to the files. If the file with the number 1 exists, ENFORM directs output to this file only. If this file does not exist, ENFORM directs output to the file with the number 2, and so on.

SECTION 3

ENFORM LANGUAGE ELEMENTS

An ENFORM query is built with elements of the ENFORM language. This section contains:

- An explanation of the ENFORM language elements that are used throughout query specifications. These elements are: reserved words, special characters, and comments.
- A brief explanation of the functions of the ENFORM statements, clauses, and commands. The syntax of these language elements is described later in this manual.
- The syntax and functions of the ENFORM language elements that can be used either in a target-list or in request-qualification. (Both target-lists and request-qualification are defined in Section 1). These language elements include aggregates, literals, arithmetic expressions, logical expressions, IF/THEN/ELSE expressions, parameters, user variables, and user tables.
- The rules to be used when referencing data base records, fields, and primary keys, when including subscripts, and when naming using defined elements such as user variables, user tables, user aggregates, or parameters.

Figure 3-1 shows a query specification and some of the language elements discussed in this section.

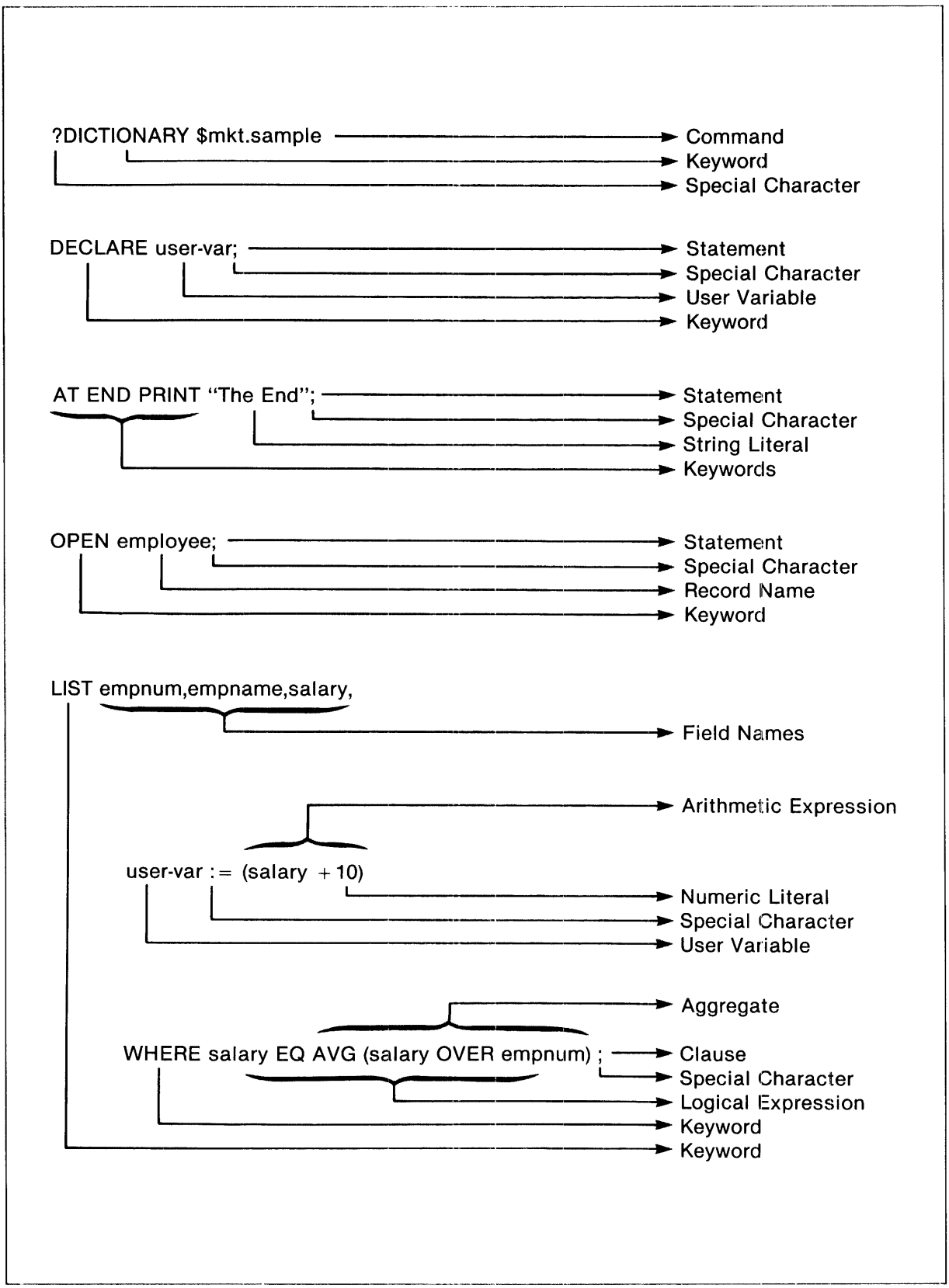


Figure 3.1. ENFORM Language Elements

RESERVED WORDS AND KEYWORDS

Reserved words are words with special meaning to ENFORM. Both in syntax and elsewhere in this publication reserved words are shown in uppercase characters. Reserved words must be spelled exactly as shown.

Do not use reserved words to name records, fields, variables, tables, or parameters. Reserved words can be redefined or translated to a language other than English. Refer to the ENFORM Users Guide for instructions on redefining the ENFORM reserved words. Table 3-1 shows the ENFORM reserved words.

Table 3-1. ENFORM Reserved Words

ACROSS *	DESC	LIST	SET	WITH
AFTER	DICTIONARY	LT	SKIP	WITHOUT
ALL	ELSE	MAX	SPACE	ZERO
AND	END	MIN	START	ZEROS
AS	EQ	NE	SUBFOOTING	'
ASCD	EQUAL	NOHEAD	SUBTITLE	(
AT	EXIT	NOPRINT	SUBTOTAL	)
AVG	FILE *	NOT	SUM	*
BEFORE	FIND	NULL	SUPPRESS	+
BEGINS	FOOTING	OF	TAB	–
BLANK	FORM	OFF	THAN	.
BLANKS	GE	ON	THEN	/
BY	GREATER	OPEN	THRU	;
CENTER	GT	OPTION	TIME	<
CHANGE	HEADING	OPTIONAL	TIMESTAMP-DATE	=
CLOSE	IF	OR	TIMESTAMP-TIME	>
CONTAINS	INTERNAL	OVER	TITLE	@
COPY	INVOKE *	PARAM	TO	[
COUNT	IS	PCT	TOTAL	]
CUM	JULIAN-DATE	PRINT	UNIQUE	
DATE	KEY	RECORD	USING *	
DECLARE	LE	ROW-SUBTOTAL *	VIA	
DEFINE *	LESS	ROW-TOTAL *	WHEN *	
DELINK	LINK	SAVE	WHERE	

* These words are reserved for future extensions to ENFORM.

Keywords are reserved words that indicate the beginning of statements or clauses to the ENFORM compiler.

SPECIAL CHARACTERS

Table 3-2 shows the ENFORM special characters and describes their functions within a query.

Table 3-2. Special Characters

Character Name	Character	Description
blank		Separates keywords and other language elements.
quotation mark	"	Serves as delimiter for various language elements, such as alphanumeric literals, and some display formats.
assignment syntax	:=	Assigns a value to a target-item.
apostrophe	'	Serves as a delimiter for display formats, some conditional operators.
parenthesis	()	Delimits various language elements. Must appear in balanced pairs.
brackets	[]	Delimits subscripts, modifiers, and decorations. Must appear in balanced pairs.
question mark	?	Denotes commands when directly followed by a keyword.
comma	,	Separates multiple query specifications in the same statement; always optional.
semicolon	;	Terminates statements

COMMENTS

A comment clarifies and documents the purpose of the your query. A comment is denoted by the exclamation character (!). A comment can be the only text on a line, the last text on a line, or text embedded within a line. When a comment is embedded within a line, it must be enclosed with exclamation marks. For example, consider the following comments:

```
!This query produces Finance Report 301

DICTIONARY finance.subvol; ! Information is confidential

OPEN finance1, !Only two files are needed! finance2;
```

STATEMENTS

Statements contain specifications for selecting and formatting elements from your data base. Composed of keywords, clauses, and target-lists, the ENFORM statements LIST and FIND provide the basic specifications for information selection. Additional statements establish the query environment and provide some report structuring capability.

With the exception of the LIST and FIND statement, ENFORM statements remain in effect (unless cancelled, reset, or overridden) for the duration of an entire ENFORM session. ENFORM requires the LIST and FIND statements to be terminated with a semicolon. The other ENFORM statements should be terminated with a semicolon since ENFORM does not report errors until it encounters either a terminating semicolon or the beginning of a new statement.

Refer to Section 4 for the syntax of the ENFORM statements.

CLAUSES

Clauses are optional elements of ENFORM statements. With the exception of the option variable clauses and the system variable clauses, ENFORM clauses only apply to the LIST or FIND statement of which they are a part.

Some of the operations performed by ENFORM clauses are:

- Sorting and grouping target-records.
- Calculating subtotals, totals, percentages, and running totals.
- Printing user supplied information within a report.
- Formatting a report.
- Extracting the current date, time, line number, and page number.
- Converting data to internal or display format.

Refer to Section 5 for the syntax of the ENFORM clauses.

COMMANDS

Commands are compiler directives that tell the compiler/report writer to perform a specific action. For example, commands tell the compiler/report writer to:

- Associate a new physical file with a record description.
- Attach a specific query processor.
- Enter the Tandem text editor without leaving ENFORM.
- Compile a program and save it in a compiled query file.
- Compile and execute an EDIT file containing source code.
- Execute a compiled query file.
- Include part of an EDIT file in the input to ENFORM.
- Display information about the current ENFORM environment.

Refer to Section 6 for the syntax of the ENFORM commands.

RULES FOR NAMING USER DEFINED ELEMENTS

When you name variables, tables, aggregates or parameters, the name:

- Must be unique.
- Must start with either an alphabetic character or a circumflex (^).
- Can contain numbers, hyphens (-), or circumflexes (^).
- Can be from 1 to 31 characters in length.
- Must not contain embedded blanks.
- Must not end with a hyphen (-).

RULES FOR REFERENCING DATA BASE ELEMENTS

When you reference a data base element within your query, you must follow certain rules. The rules used to reference a record name, a field name, and a primary key are described in the following paragraphs.

Record Name References

When you reference a record name within a query, the record name must be unique. If a record name is the same as a field name in an open record description, ENFORM assumes the unqualified reference refers to the field name.

Referencing a record name as a target-item is the same as referencing each occurrence of each of the fields individually. A record name cannot be specified as an element in a print list. (A print list is part of the AT END statement and clause, the AT START statement and clause, the FOOTING statement and clause, the SUBFOOTING statement and clause, the SUBTITLE statement and clause, the TITLE statement and clause, and the BEFORE CHANGE and AFTER CHANGE clauses.)

Field Name References

The same field name can exist in more than one data base file. If your query involves data base files with duplicate field names, the field name must be uniquely qualified.

Field names can be qualified by using two different conventions. The first convention joins the record or group name to the field name with a period:

```
record-name.field-name
        or
group.name.field-name
```

The second convention joins the record or group name to the field name with the keyword OF. When the OF syntax is used the field name or group name is written first followed by OF and the qualifier needed:

```
field-name OF record-name
        or
field-name OF group-name
```

A field name requires as much qualifying as necessary to uniquely identify the field to ENFORM. The necessary qualification might be as simple as combining the field name with the record name or group name. It might require combining the field name with both a group name and a record name or with two group names. Consider the record descriptions shown in Figure 3-2.

```
RECORD stock-items.                    RECORD shelf-items.
FILE IS "stock"    KEY-SEQUENCED.      FILE is "shelf"    KEY-SEQUENCED.
  02 depot-num     PIC 99.               02 dept-num      PIC 99.
  02 cont-num      PIC 99.               02 dept-name     PIC X(10).
  02 erasers.                            02 cont-num      PIC 99.
     05 ink        PIC 99.               02 pens.
     05 gum        PIC 99.                  05 b-point    PIC 99.
     05 pink       PIC 99.                  05 felt-tip   PIC 99.
  02 ink pens.                           02 erasers.
     05 felt-tip   PIC 99.                  05 ink        PIC 99.
     05 b-point    PIC 99.                  05 gum        PIC 99.
     05 fountain   PIC 99.                  05 gray       PIC 99.
```

Figure 3-2. Records With Duplicate Field Names

If both the *stock-items* and *shelf-items* record descriptions are open, *ink* must be qualified. To qualify *ink* within *shelf-items*, one of the following must be entered:

```
ink OF erasers OF shelf-items

shelf-items.erasers.ink
```

Primary Key References

The records in data base files can be uniquely identified by the value of a primary key. For data base files with key-sequenced file structure, the primary key is part of the record. For data base files with relative, unstructured, or entry-sequenced file structure, the primary key is not part of the record. Primary keys can be referenced in two forms:

```
KEY OF record-name

        or

  record-name.KEY
```

The form record-name.KEY can appear for only one relative, entry-sequenced, or unstructured file per query.

The primary key for files with key-sequenced file structure is a field within the record. For key-sequenced files, referencing the primary key using the form record-name.KEY is the same as explicitly naming the field described as the primary key. The advantage of the form record-name.KEY is that you do not have to know the name of the primary key field in order to reference it. A listing of the primary key values of the parts file can be obtained by:

```
OPEN parts;
LIST parts.KEY;
```

The record with the primary key value of 1403 can be referenced by:

```
WHERE parts.KEY = 1403
```

For files with relative file structure, the primary key is a record number. The record number is the ordinal position of the record relative to the beginning of the physical file. The first record in the physical file has position zero. A listing of the primary keys of a relative file can be obtained by:

```
OPEN rel1;
LIST KEY OF rel1;
```

The primary key of the fifth record in the file can be referenced by:

```
WHERE KEY OF rel1 = 4
```

Remember, the fifth record in a file with relative file structure has position four because the first record has position zero.

For files with entry-sequenced and unstructured file structures, the primary key is a record address. A record address is the byte offset from zero of the record you want. A record address is always an even number. A listing of the primary keys of an entry-sequenced file can be obtained by:

```
OPEN entryseq;
LIST entryseq.KEY;
```

The third record in the file entryseq has a primary key whose byte offset is 16. The record can be referenced by:

```
WHERE entryseq.KEY = 16
```

For more information about file structures, refer to the *ENSCRIBE Data Base Manager Programming Manual.*

SUBSCRIPTS

Subscripts, although they are not required, are usually used to reference elements in a user table or data base table. (A data base table is created when the dictionary description of a data base field contains an OCCURS clause.) Subscripts are needed for references to user tables and data base tables because all the elements in such tables have the same name. Subscripts can be used in references to data base fields and user variables although they are not necessary.

The syntax for including subscripts is:

```
{field-name-ref1 }  { "["subscript"]"         }
{user-table-name }  { "["subscript-range"]" }

  grp-name {"[" subscript "]"     } .field-name-ref2 {"["subscript"]"         }
           {"["subscript-range"]"}                   {"["subscript-range "]"}
```

where

`field-name-ref1`

is the qualified name of a data base field.

`user-table-name`

is the name of a user table defined by the DECLARE statement.

`subscript`

is an integer. The lowest valid value for *subscript* is 1. The highest valid value is the maximum number of elements defined for the user or data base table. (Refer to the following discussion for more information.)

`subscript-range`

has the form `subscripti : subscriptj`.

where

`subscripti`

is the first element being referenced.

`subscriptj`

is the last element being referenced.

`grp-name`

is the name of a group described in the dictionary. A group is defined as a record element whose level number (02, 03, 04,...) is less than that of the next record element.

`field-name-ref2`

is the name of a subordinate field. A subordinate field is defined as a record element whose level number (05, 06, 07,...) is greater than that of *grp-name*.

When a subscript is included with a reference to a table name, a user variable name, or a field name, ENFORM determines whether the subscript is a valid subscript value allowed for the table, variable, or field. A valid subscript value for a field or user variable is 1. Valid subscript values for a user table are defined in the DECLARE statement. For example, consider the following user table declaration:

```
DECLARE u-var[ 24 ];
```

The valid subscript values for *u-var* are 1 through 24. Valid subscript values for data base tables are defined by the OCCURS clause in the dictionary description of the table. For example:

```
02 monthly-sales OCCURS 12 TIMES.
```

The valid subscript values for *monthly-sales* are 1 through 12.

Both user and data base tables can be referenced without a subscript. ENFORM assumes a subscript of 1. For example:

`u-var`	Refers to the first element of the user table.
`monthly-sales`	Refers to the first element of the data base table.

Including a subscript with a user or data base table reference identifies the individual elements of the table. For example:

`u-var [ 2 ]`	Refers to the second element of the user table.
`monthly-sales [ 4 ]`	Refers to the fourth element of the data base table.

Subscript-range can be included in user or data base table references when the table is used as a target-item in a LIST statement. *Subscript-range* is illegal if a data base table is modified by a BY, BY DESC, ASCD, or DESC clause. (Refer to Section 5 for information about these clauses.) Including *subscript-range* is the same as referencing the table elements individually. For example:

`u-var [ 3:8 ]`	Refers to the third through eighth elements of the user table.
`monthly-sales [ 1:4 ]`	Refers to the first through fourth elements of the data base table.

In the dictionary record description, a data base table can be defined as a group element with subordinate data base field entries. For example:

```
02 sales   OCCURS 12 TIMES.
   10 month        PIC X(3).
   10 top-dept     PIC 9999.
```

ENFORM allows you to refer to the subordinate data base fields as follows:

`sales`	Refers to *month* and *top-dept* within the first element of *sales*.
`sales [ 2 ].top-dept`	Refers to *top-dept* within the second element of *sales*.
`sales [ 2:3 ].month`	Refers to *month* within the second through third elements of *sales*.

The subordinate elements of a data base table can themselves contain a data base table resulting in nested data base tables. For example, consider the following record description:

```
02 tot-sales       OCCURS 12 times.
   10 month        PIC X(3).
   10 top-dept     PIC 999.
   10 wkly-sales   PIC 99999 OCCURS 4 TIMES.
```

ENFORM allows you to reference nested data base tables. For example:

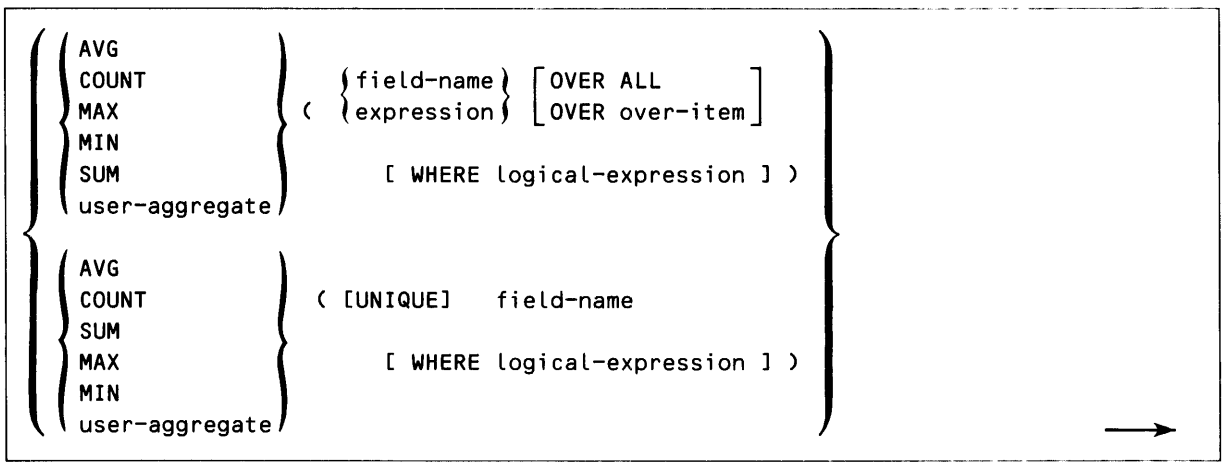
`tot-sales`	Refers to *month, top-dept, and* the first element of wkly-sales within the first element of *tot-sales.*
`tot-sales[4].wkly-sales[2]`	Refers to the second element of *wkly-sales* within the fourth element of *tot-sales.*
`tot-sales[1:5].wkly-sales[3]`	Refers to the third element of *wkly-sales* within the first thru fifth elements of *tot-sales.*
`tot-sales[2:3].wkly-sales[1:4]`	Refers to the first thru fourth elements of *wkly-sales* within the second thru third elements of *tot-sales.*

AGGREGATES

An aggregate is the result of a cumulative operation performed for each value that contributes to the aggregate. An aggregate yields a single value for the group of values over which it is processed. Aggregates can only be specified as a target-item or in a request-qualification. Aggregates specified as a target-item are called target aggregates. Aggregates specified in a request-qualification are called qualification aggregates.

ENFORM provides two types of aggregates: predefined and user-defined.

The syntax used for an aggregate is:



where

AVG

is a predefined ENFORM aggregate that computes an average value for a set of numbers or expressions.

COUNT

is a predefined ENFORM aggregate that tallies the occurrences of a field defined as either numeric or alphanumeric.

MAX

is a predefined ENFORM aggregate that finds the highest number in a set of numbers or expressions, or finds the alphanumeric string with the highest value based on the ASCII collating sequence.

MIN

is a predefined ENFORM aggregate that either finds the lowest number in a set of numbers or expressions or finds the alphanumeric string with the lowest value based on the ASCII collating sequence.

SUM

is a predefined ENFORM aggregate that totals a set of numbers or expressions.

user-aggregate

is the name of an aggregate defined by a DECLARE statement. (See Section 4).

field-name

is the name of a data base field.

expression

is a arithmetic or IF/THEN/ELSE expression (explained later in this section.)

over-item

is a field used to sort and group the records over which the aggregate is processed. For a target aggregate, *over-item* must be a by-item (the name of a field modified by a BY or BY DESC clause). For a qualification aggregate, *over-item* can be any field name.

OVER ALL

defines the range of the aggregate operation as over all the values specified. →

OVER

defines a range for the aggregate operations. The aggregate operation takes place only over the specified *over-item.* The operation yields one aggregate value for each unique value of *over-item.*

UNIQUE

excludes duplicate values from contributing to the collecting operation of the aggregate. UNIQUE adds considerable processing overhead and should not be specified unless you know unwanted duplicate values exist. *UNIQUE* is redundant with *MAX* or *MIN.* While the same answer is returned when *UNIQUE* is specified with these aggregates, processing time increases greatly.

Predefined Aggregates

The five predefined ENFORM aggregates are: AVG, COUNT, MAX, MIN, and SUM.

AVG finds an average value for a set of numbers. For each record containing a field value to be averaged, the field value is added to the running total of the contributing field values. After all contributing field values are processed, the final total is divided by the number of field values that made up the total. The following example finds the average of the partcost field over the suppnum field:

```
LIST BY suppnum,
  AVG(partcost OVER suppnum);
```

COUNT tallies the instances of an element. In the following example, the number of parts kept in stock are counted by counting the part numbers in *parts:*

```
OPEN parts;
LIST COUNT(partnum);
```

MIN determines the lowest number in a set of numbers or expressions. MAX determines the highest number in a set of numbers or expressions. The following example finds both the lowest and the highest price of a part:

```
OPEN fromsup;
LIST BY partnum,
  MIN(partcost OVER partnum),
  MAX(partcost OVER partnum);
```

SUM totals a set of numbers or expression values. In the following example, the sum of the sales made by each salesman is obtained:

```
OPEN order,odetail,parts;
LINK order to odetail VIA ordernum;
LINK odetail to parts VIA partnum;
LIST BY salesman,
  SUM (price * quantity) OVER salesman);
```

User Aggregates

When the predefined aggregates do not meet your needs, you can define your own aggregates with a DECLARE statement. A user-defined aggregate can be used anywhere a predefined aggregate can appear with two exceptions:

- A user aggregate cannot be referenced in the end expression of the declaration of another user aggregate.
- A user aggregate declared with an end expression cannot be used as a qualification aggregate OVER a by-item.

The syntax of a user-defined aggregate is shown in Section 5 with the DECLARE statement. The syntax is also shown here to clarify this discussion.

```
user-aggregate-name ( formal-argument ) = ( step-expression
  [ , [ end-expression ] [ ,initialize-constant ] ] )
```

where

`user-aggregate-name`

is a unique name you give your aggregate. The name must follow the naming rules described earlier in this section. This name is also used to obtain the current value of the aggregate in *step-expression* and *end-expression.*

`formal-argument`

is a unique name used to represent the actual field name or expression in the aggregate definition.

`step-expression`

is an arithmetic expression to be computed for each value contributing to the aggregate. Normally *step-expression* includes a reference to the *user-aggregate-name* so that a value is accumulated over the contributing values.

`end-expression`

is an arithmetic expression to be computed after all qualifying values for the aggregate are processed. *End-expression* can contain a pre-defined aggregate name. If *end-expression* is omitted and *initialize-constant* is present, an extra comma must precede *initialize-constant.*

`initialize-constant`

is a numeric literal that is the starting value of the aggregate. Zero is the default. When *end-expression* is omitted and *initialize-constant* is present, an extra comma must precede *initialize-constant.*

Parentheses must appear exactly as shown in this syntax.

Consider the following user aggregate:

```
DECLARE grossavg (x) = (grossavg = .10 * x, grossavg/COUNT (x));
```

The user-aggregate name is grossavg. The *formal-argument* is x. The step expression is (grossavg + .10 * x). The *end-expression* is grossavg/COUNT (x).

This user-aggregate could be used to compare the new gross salaries to the old average salary:

```
LIST AVG (salary),
     grossavg (salary);
```

Target Aggregates

A target aggregate appears as part of the ENFORM output. To specify a target aggregate, follow these rules:

- The field names in the target aggregate syntax can come from different data base records.
- *Over-item* must be a by-item.
- Target aggregates cannot be nested; that is, one aggregate cannot be used as the argument for another aggregate.

TARGET AGGREGATE WITH OVER ALL SYNTAX. When you specify a target aggregate without specifying either *OVER* or *OVER ALL*, ENFORM assumes *OVER ALL*. Figure 3-3 shows both a query outline using the *OVER ALL* syntax and an output diagram.

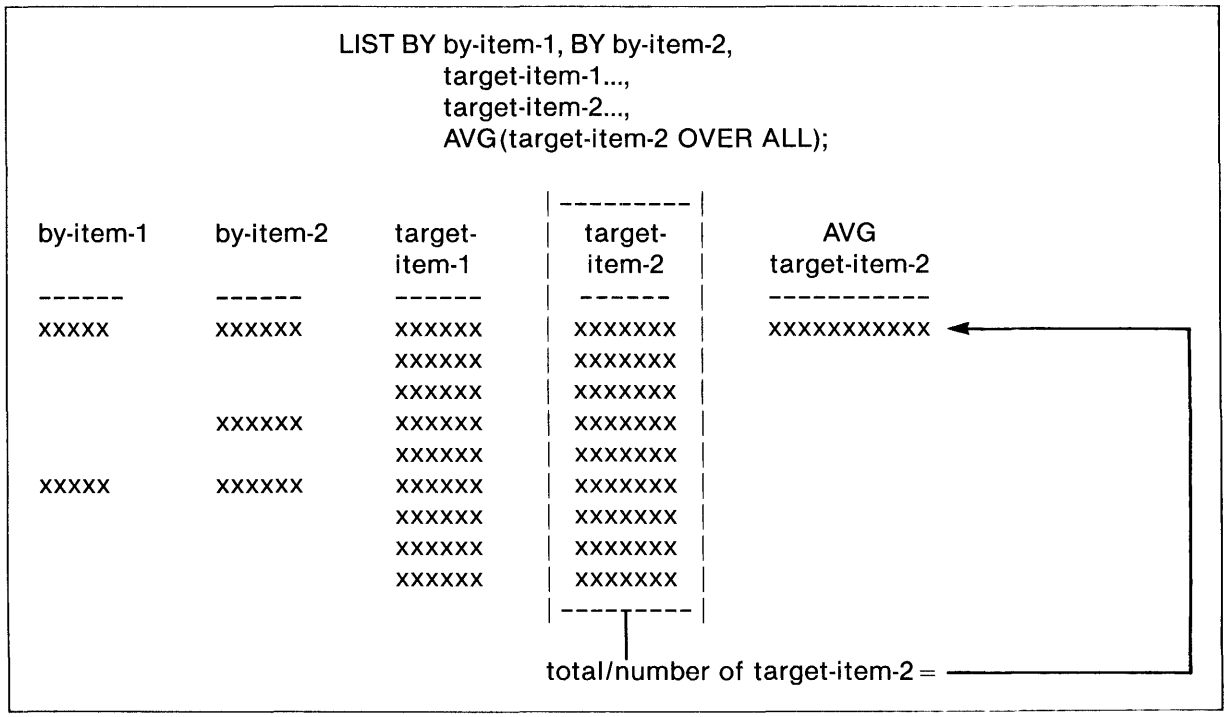


Figure 3-3. Query Outline of Target-Aggregate With OVER ALL Syntax

Specifying the aggregate shown in Figure 3-3 is the same as specifying:

```
AVG(target-item-2);
```

The query groups target-item-1 and target-item-2 within by-item-2. The aggregate AVG is used and the average value of target-item-2 is found by totaling all of the values of target-item-2 and dividing that total by the number of target-item-2 values. Notice that AVG target-item-2 prints as a separate column on the report with only one non-blank entry. This entry corresponds to the aggregate value of all records in the report.

When a target aggregate is specified in a FIND statement with OVER ALL either present or assumed, only the first target-record contains the aggregate value. This field is blank in all other target-records generated by the FIND statement.

TARGET AGGREGATE WITH OVER SYNTAX. If you specify a target aggregate in a LIST or FIND statement with the OVER syntax, a single aggregated value is returned for each grouped value. Consider the query outline and output diagram shown in Figure 3-4.

Query outline:

```
LIST BY by-item-1,
     BY by-item-2,
        target-item-1
     AVG(target-item-1 OVER by-item-2);
```

Output Diagram:

```
                               |---------|
by-item-1      by-item-2       | target- |            AVG
                               |  item-1 |       target-item-1
------         ------          |---------|       -----------
xxxxxxx        xxxxxxx         | xxxxxxx |       xxxxxxxxxxx
                               | xxxxxxx |   ------> (total/4)
                               | xxxxxxx |
                               | xxxxxxx |
                               |---------|
               xxxxxxx         | xxxxxxx |       xxxxxxxxxxx
                               | xxxxxxx |   ------> (total/2)
                               |---------|
xxxxxxx        xxxxxxx         | xxxxxxx |       xxxxxxxxxxx
                               | xxxxxxx |   ------> (total/3)
                               | xxxxxxx |
                               |---------|
```

Figure 3-4. Query Outline of Target-Aggregate with OVER Syntax

In this example, AVG target-item-1 prints as a separate report column with a non-blank entry only on the first line of each new by-item-2 group. This entry represents the average of target-item-1 over the by-item-2 group.

When a target aggregate is specified in a FIND statement with the OVER syntax, the aggregate value is present only in the first target-record for a by-item. This field is blank for all other target-records.

QUALIFICATION AGGREGATES. Qualification aggregates must be specified in a WHERE clause. When specifying a qualification aggregate, follow these rules:

- Qualification aggregates cannot be nested; that is, one aggregate cannot be used as the argument for another aggregate.
- A qualification aggregate can contain an embedded WHERE clause.
- The embedded WHERE clause can contain another qualification aggregate; however, that qualification aggregate must be computed OVER ALL.
- All field names used for each individual qualification aggregate expression (the field being aggregated, any field names appearing in an expression being aggregated, and any field names appearing in the WHERE clause associated with an aggregate) must come from the same data base record.
- The *UNIQUE* syntax cannot be used with a qualification aggregate.

QUALIFICATION AGGREGATE WITH OVER ALL SYNTAX. When you specify a qualification aggregate without specifying either *OVER* or *OVER ALL*, ENFORM assumes *OVER ALL*. ENFORM computes the value for the qualification aggregate over all the records in the original data base (not over the target-records). For example:

```
LIST odetail,
  WHERE quantity > AVG (quantity );
```

restricts the records returned from *odetail* to those whose quantity field is greater than the average of all the values of the quantity field.

QUALIFICATION AGGREGATES WITH OVER SYNTAX. When a qualification aggregate is used with the *OVER* syntax, ENFORM computes one value for each *over-item* group. When the WHERE clause is evaluated, ENFORM uses the qualification aggregate value for the particular group to restrict the records selected. Figure 3-5 shows a query outline and the process that occurs when the *OVER* syntax is used.

```
OPEN  employee;
LIST  regnum,
      branchnum,
   WHERE salary GT AVG (salary OVER regnum);
```

The qualification aggregate value for each group is:

regnum	AVG salary
1	24666
2	28333
5	38000
99	39500

If employee has the following form when grouped by regnum and branchnum:

regnum	branchnum	salary
1	1	36000
		19000
		25000
		26000
		12000
	2	30000
2	1	37000
		25000
		23000
5	3	38000
99	1	39500

The report produced is:

Region	Branch
2	1
1	1
1	1
1	1
1	2

Figure 3-5. Query Outline of Qualification Aggregate With OVER Over-item Syntax

QUALIFICATION AGGREGATE WITH AN EMBEDDED WHERE CLAUSE. If a qualification aggregate contains a WHERE clause that restricts the records for the aggregate calculation, ENFORM processes the embedded WHERE clause before the aggregate. If any record for an *over-item* does not satisfy the restriction specified in the embedded WHERE clause, ENFORM excludes all records for that *over-item* from the aggregate calculation.

For example consider Figure 3-6. This figure shows an ENFORM query containing a qualification aggregate with a WHERE clause. The query prints the part number and the amount in stock of all the parts where the price of the part is greater than the average price of all parts not in stock.

```
OPEN parts;
LIST  partnum,
      inventory,
 WHERE price GT AVG (price WHERE inventory LT 0);
```

The report produced is:

Part Number	INVENTORY
212	7
244	3
1403	21
5502	6
5504	-1
5505	0
7102	20

Figure 3-6. Qualification Aggregate With Embedded WHERE Clause

LITERALS

Literals can be used in both a target-list and a request-qualification. Literals can also be used in many ENFORM statements and clauses. Literals are used in titles, headings,special text printed within a report's body, and in expressions. The two types of literals are numeric and string.

Literals cannot be continued across lines. The maximum length of a literal is 127 characters.

Numeric Literals

Numeric literals are used in all arithmetic expressions. They can be used in logical expressions when the literal is compared to a data base element described in the data dictionary as numeric. Numeric literals:

- Are not enclosed in quotation marks.
- Are composed of the digits 0-9.
- Cannot be larger than 32767.

- Can be preceded or followed by a plus or minus sign.
- Must be enclosed in parentheses if they are specified outside of a logical expression or a TAB, SPACE, SKIP, or FORM clause.

Numeric literals can stand alone as target-items in a LIST or FIND statement. In this case they must be enclosed in parentheses.

The following are examples of numeric literals:

```
(104) (123.0444) (+267) (.006) (-15)
```

String Literals

String literals can be used in many of the ENFORM statements and clauses. String literals can be used in logical expressions if the data base element to which the string literal is compared is declared alphabetic or alphanumeric in the data dictionary. String literals:

- Can be composed of any character in the ASCII character set.
- Must be enclosed in quotation marks. If a quotation mark is part of a string literal, the quotation mark must be doubled.

The following are examples of string literals:

```
"This is a string literal"

"This string literal contains a " " quotation mark"

"1234.99"
```

String literals can stand alone as target-items in a LIST statement. Using a string literal in this manner allows printing of one or more constant characters between two columns of data. For example:

```
LIST customer,"....",address;
```

produces the following report:

```
   CUSTOMER                    ADDRESS
------------------    --------------------
CENTRAL UNIVERSITY....UNIVERSITY WAY
BROWN MEDICAL CO  ....100 CALIFORNIA STREET
```

ARITHMETIC EXPRESSIONS

Arithmetic expressions are some combination of numeric literals, field values, variables, or aggregates that are added, subtracted, multiplied, or divided to yield a single value. A JULIAN-DATE clause, TIMESTAMP-DATE clause, or a TIMESTAMP-TIME clause can also be used in an arithmetic expression. Arithmetic expressions must be enclosed in parentheses.

Table 3-3 shows the arithmetic operators and their functions.

Table 3-3. Arithmetic Operators

Operator	Function
+	Addition
–	Subtraction
*	Multiplication
/	Division

Spaces are not required before any of the arithmetic operators with the exception of the subtraction sign (–). At least one space must precede a subtraction sign that follows a field or variable name.

Arithmetic expressions can be simple:

```
(price + 10.00)
```

or they can be complex: :

```
((price + 10.00)* quantity)
```

Evaluation Order of Arithmetic Expressions

Arithmetic expressions are evaluated in this order:

1. Nested parenthesized expressions are evaluated first, beginning with the innermost expression.

2. Within a nested parenthesized expression, multiplication and division operations are evaluated next.

3. Within a nested parenthesized expression, addition and subtraction operations are evaluated last.

Scale Factor of the Result

The scale of the result is determined by the number of digits after the decimal point. In an arithmetic expression, the result has the same number of digits after the decimal point as the field or variable in the expression with the greatest precision. This could result in loss of significant digits if too great a precision is used for the field or variable.

The resulting scale factor can be controlled by assigning the result of the arithmetic expression to a user variable. (User variables are explained later in this section.) The precision of the user variable can be specified by an INTERNAL clause within the DECLARE statement that defines the user variable. The maximum number of digits allowed is 18. All calculations with ENFORM are performed with QUAD arithmetic.

LOGICAL EXPRESSIONS

Logical expressions evaluate to a truth value—either true or false based on a condition specified within the expression. Both the conditions that can be specified and the conditional operators are shown in Table 3-4.

Table 3-4. Conditional Operators

Condition	Keyword	Abbreviation	Symbol
Equal	EQUAL IS	EQ	=
Not equal		NE	<>
Greater than	GREATER [THAN]	GT	>
Greater than or equal to		GE	>=
Less than	LESS [THAN]	LT	<
Less than or equal to		LE	<=

ENFORM provides two other conditional operators: BEGINS WITH and CONTAINS. The three symbols ']' are synonymous with BEGINS WITH and the three symbols '>' are synonymous with CONTAINS.

The syntax of a logical expression is:

```
[NOT] condition [ {AND} [NOT] condition ] ...
               [ {OR }                  ]
```

where

condition

has one of the following forms:

```
                 [ not ] {BEGINS WITH  }
                         {  ']'        }
                         {CONTAINS     }
                         {  '>'        } string-literal
                         {conditional operator}
field-name {
                 [ NOT ] {EQUAL}
                         {EQ   }
                         {IS   }                 {value-range          }
                         {=    }                 {"["pattern-match"]"  }
                         {NE   }
                         {< >  }
           }

{variable  }
{field-name}  [NOT] conditional-operator {variable  }
{expression}                             {field-name}
                                         {expression}
```

where

conditional-operator

is one of the conditional operators shown in Table 3-4.

expression

is an IF/THEN/ELSE or arithmetic expression.

pattern-match

is a pattern of numbers or characters, enclosed within both quotation marks and brackets. The syntax for a *pattern-match* is:

```
{"["  n                          n     "]"}
{     m, n  string-literal ...   m, n      }
{     -                          -       }
```

→

where

n

is an integer indicating that exactly *n* number of characters must precede or follow *string-literal* when it is found in a field value.

string-literal

is the pattern of characters or numbers to which the field is being compared. *String-literal* must be enclosed in quotation marks.

m,n

is two integers separated by comma indicating that at least m characters but not more than n characters must precede or follow *string-literal* when it is found in a field value.

-

is a dash indicating any number of characters (0 thru 255) can precede or follow *string-literal* when it is found as a field value. Specification of a dash indicates you do not care about the contents of this part of the field.

value-range

is a range of values with the form: value-1 THRU value-2

A logical expression can be simple or compound. A simple logical expression consists of one condition. A compound logical expression uses the boolean operators *AND*, *OR*, and *NOT* to operate over two or more logical expressions.

Using the boolean operators *AND*, *OR*, and *NOT* has the following effect on the evaluation of the logical expression:

- When you precede a condition with the boolean operator *NOT*, the result of the expression is evaluated as true if the condition is evaluated as false.
- When you join two or more conditions with the boolean operator *AND*, the result of the expression is evaluated as true only if all the conditions are evaluated as true.
- When you join two or more conditions with the boolean operator *OR*, the result of the expression is evaluated as true if any of the conditions are evaluated as true.

Effect of Parentheses on Compound Logical Expressions

Compound logical expressions are evaluated in this order:

1. Conditions within parentheses are evaluated first.
2. Conditions preceded by the boolean operator *NOT* are evaluated second.
3. Conditions joined with the boolean operator *AND* are evaluated third.
4. Conditions joined with the boolean operator *OR* are evaluated last.

BEGINS WITH and CONTAINS

BEGINS WITH and *CONTAINS* are special conditional operators that can be used to yield a true or false value if a field either begins with or contains a specific alphanumeric string.

The *BEGINS WITH* operator determines if a field starts with a specified alphanumeric string. For example, the following can be used to limit the data retrieved from the supplier field to only those records whose suppname field begins with TANDEM:

```
LIST supplier WHERE suppname BEGINS WITH "TANDEM",
```

In this example, the logical expression is evaluated as true only if the field begins with a value where TANDEM is in uppercase characters.

The *BEGINS WITH* operator can only be used with fields specified as alphanumeric in the data dictionary. A field is alphanumeric when its corresponding data description entry is specified as PIC X.

The *CONTAINS* operator determines if a field contains a specified alphanumeric string. For example, the following determines whether the empname field contains the value GEORGE:

```
empname CONTAINS "GEORGE"
```

Range of Values in Logical Expressions

A field can be compared to a range of values in a logical expression. In the range, value-1 must be less than value-2. Values used in a range can be either numeric literals or string literals.

Specifying field EQ value1 THRU value2 is equivalent to specifying:

```
field-name GE value-1 AND field-name LE value-2
```

A range can contain a numeric literal if the field being examined is defined as numeric in the data dictionary. The inventory field of the parts file is defined as PIC 999. The following logical expression is evaluated as true if the value of the inventory field falls within the range of 5 to 15, including the values 5 and 15:

```
inventory EQ 5 THRU 15,
```

A range expression can also use a string literal if the field being examined is defined as alphanumeric in the data dictionary. The partname field is defined as PIC X(18). The following logical expression is evaluated as true if partname field contains a value that falls with in the range of A to L:

```
partname EQ "A" THRU "LZZZZZZZZZZZZZZZZZ"
```

Note that the second string literal is 18 characters long, the length of the partname field. Specifying the literal in this manner ensures that all of the part names beginning with L are included.

Pattern-Match in Logical Expressions

In a logical expression a field described as alphanumeric in the data dictionary can be compared to a *pattern-match*. A *pattern-match* is actually a comparison template that the field value is compared to. In the following logical expression, the partname field is compared to a *pattern-match*:

```
partname = [-"DISK" - "MB"-],
OR partname = [-"DISK"-"Mb"-],
OR partname = [-"Disk"-"MB"-],
OR partname = [-"Disk"-"Mb"-],
```

In the following logical expression, two numbers precede the *string-literal* indicating that at least one character but no more than two characters must precede *string-literal* in the field value:

```
partname EQ [1,2 "T"-],
```

IF/THEN/ELSE EXPRESSIONS

IF/THEN/ELSE expressions yield a value determined by the result of a logical expression. IF/THEN/ELSE expressions can be used wherever an arithmetic expression can be used. IF/THEN/ELSE expressions can be nested. The syntax of an IF/THEN/ELSE expression is:

```
(IF logical-expression THEN value-1 ELSE value-2)
```

where:

logical-expression

is a logical expression that evaluates as true or false.

value-1 or value-2

is a field name, an arithmetic expression, or IF/THEN/ELSE expression, or one of the following value keywords: NULL, BLANK, BLANKS, ZERO, ZEROS.

If the logical expression is evaluated as true, *value-1* is used. If the logical expression is evaluated as false, *value-2* is used. *Value-1* and *value-2* must be the same data type, either both numeric or both alphanumeric.

The value keywords NULL, BLANK, and BLANKS print blanks on reports. The value keywords ZERO and ZEROS print zeros on reports.

Consider the following IF/THEN/ELSE expression:

```
IF partnum = 2001 THEN ZEROS ELSE partnum,
```

This expression specifies that if partnum is equal to 2001, then zeros are to be printed on the report. If partnum has any other value except 2001, partnum prints.

PARAMETERS

You can use a parameter either for request qualification or as a target-item. You can pass a parameter to a stored compiled query file. Define parameters by issuing the PARAM statement, described in Section 5.

ENFORM handles parameters syntactically as if they were literals. ENFORM handles any parameter declared with an alphanumeric internal format as a string literal. ENFORM handles all other parameters as numeric literals. You must enclose a parameter with parentheses wherever you would have to enclose a numeric literal with parentheses.

USER VARIABLES

A user variable can be used to store numeric or string literals, save a field value, or hold the result of a calculation for later printing.

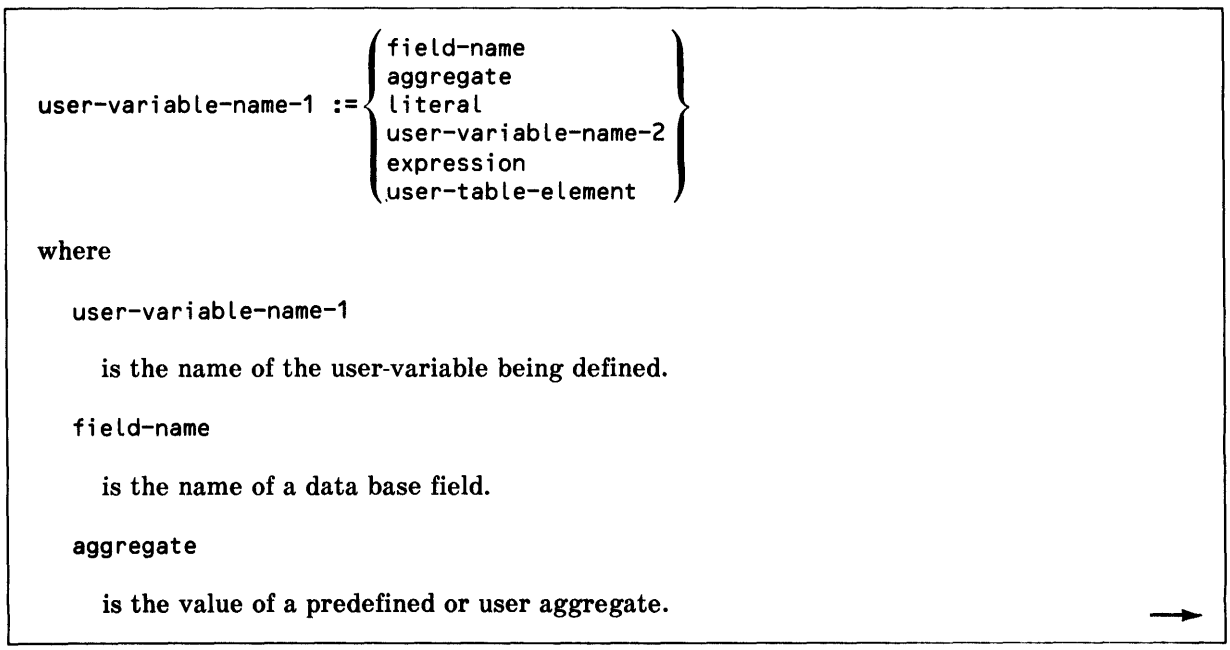
Before the user variable is specified in a query, the DECLARE statement (see Section 4 for the syntax of the DECLARE statement) must be entered. This statement defines the variable name and optionally defines the internal storage format (the default internal storage format is a 64-bit signed integer), a default display format, and a default heading. The name given to the user variable must conform to the naming conventions described in this section.

The default value of a user-declared-variable is zero. An initial value for the user variable can be defined with the SET statement.

User Variable as a Target-Item

When a user variable is specified as a target-item, ENFORM uses the default value or the initial value, whichever is appropriate. When a user variable is a target-item in a LIST statement, assignment syntax can be used to specify a new value for the user variable. The value of a user variable changes as target list elements are evaluated, so that at any time, the value of the user variable depends upon the value most recently assigned.

Assignment syntax is:



where

user-variable-name-1

is the name of the user-variable being defined.

field-name

is the name of a data base field.

aggregate

is the value of a predefined or user aggregate.

→

`literal`

is a numeric or string-literal that agrees in type with the user variable.

`user-variable-name-2`

is the name of a previously defined user variable.

`expression`

is an arithmetic or IF/THEN/ELSE expression.

`user-table-element`

is the subscripted name of a user table element. You cannot assign a subscript range to a user variable.

When assignment syntax is used, ENFORM reassigns a value to the user variable for each target-record; therefore, the value of the user variable might be different for each target-record. For example:

```
LIST u-var, u-var := salesman;
```

ENFORM uses the default or initial value for the first occurrence of *u-var* in every target-record. For the second occurrence of *u-var* ENFORM uses the value of the salesman field. This value changes for every target-record.

A user variable can be assigned the value of an expression that contains the user variable. For example:

```
DECLARE u-var;
SET u-var TO 10;
OPEN parts;
LIST partname,
     u-var:= (u-var +10);
```

ENFORM uses the initial or default value of the user variable to determine the value of the expression. In the example the value of the expression is 20. ENFORM then assigns this value to the user variable. Within the same LIST statement, assignment syntax can subsequently be used to assign the user variable to another expression containing the user variable. For example:

```
DECLARE u-var;
SET u-var TO 10;
OPEN parts;
LIST partname,
     u-var:= (u-var +10),
     u-var:= (u-var +20);
```

ENFORM uses 20, the value assigned in the first assignment syntax (*u-var + 10*) for the value of *u-var* in the second expression. After determining the value of the expression ((*u-var +10*) *+20*)), ENFORM assigns the value of the expression (40) to the user variable.

ENFORM performs this process for every target-record. ENFORM continues the process of reevaluating the value of a user variable until it encounters the end of the target-list.

A User Variable in Request-Qualification

A user variable can be specified in a request-qualification. When a user variable is used for request qualification, ENFORM always uses either the default or initial value whichever is appropriate. Consider the following:

```
SET u-var to 10;
LIST ordernum,
     uvar:= (ordernum + u-var)
   WHERE uvar > 10;
```

The preceding query always returns zero target-records even though in every target-record the value of u-var (ordernum + 10) is greater than 10. No target-records are returned because ENFORM uses the initial value of the user variable to evaluate the WHERE clause. Since u-var is set to 10, no target-records are selected for a WHERE clause with a logical expression u-var > 10.

User Tables

User tables are special kinds of user variables that can store more than one value. Currently a user table can have a maximum of 64 elements called occurrences. ENFORM issues an error message if you attempt to define a table with more than 64 occurrences.

The individual elements of a user table can be referenced by using subscripts. Refer to the discussion of subscripts in this section for more information.

Like user variables, user tables can be initialized by the SET statement. The default value of all elements in a user table is zero. When a user table is specified as a target-item, ENFORM determines its value in the same manner the value of a user variable is determined. Assignment syntax can be used to assign values to single elements of a table.

SECTION 4

STATEMENTS

This section contains the syntax of the ENFORM statements. The statements are arranged in alphabetical order to provide ease of access.

The ENFORM statements, with the exception of the LIST and FIND statement, have a session-wide affect unless cancelled or overridden. The LIST and FIND statements effect only the queries of which they are a part.

The AT END, AT START, FOOTING, SUBFOOTING, SUBTITLE, and TITLE statements apply only to queries containing a LIST statement. These statements supply information to be printed in a report.

The LIST and FIND statements must be terminated with a semicolon. The other ENFORM statements should be terminated with a semicolon. ENFORM neither executes the statement nor reports any syntax errors until it encounters either a terminating semicolon or the keyword indicating the start of the next statement; therefore, if you enter a ?SHOW command after a statement without a terminating semicolon, the effect of the statement is not shown in the output produced by the ?SHOW command.

Table 4-1 shows the ENFORM statements and their functions.

Table 4-1. Summary of Statements

Statement	Function
	Information Selection
LIST	specifies the information selected for a report and prints the report.
FIND	specifies the information retrieved from the data base and either writes the information to a physical file or transmits the information to a host language program.
	Query Environment
CLOSE	deletes a user variable, aggregate, or table, a parameter, or a record description from the internal table.
DECLARE	defines a user variable, user aggregate, or user table.
DELINK	clears a connecting relationship between record descriptions.
DICTIONARY	identifies the subvolume containing a dictionary. It also clears the internal table and reclaims table space.
EXIT	terminates the current ENFORM session.
LINK	specifies a connecting relationship between record descriptions.
OPEN	accesses a record description.
PARAM	names and defines a parameter that can receive a value from a Command Interpreter PARAM command.
SET	initializes a user variable, user table, or a parameter and resets option variables.
	Report Information Formatting
AT END	prints information at the end of all subsequent reports in the current session. See also the AT END PRINT clause in Section 5.
AT START	prints information just before the first set of column headings for all subsequent reports in the current session. See also the AT START PRINT clause in Section 5.
FOOTING	prints a footing at the bottom of each report page for all subsequent reports in the current session. See also the FOOTING clause in Section 5.
SUBFOOTING	prints a subfooting at the bottom of each report page for all subsequent reports in the current session. See also SUBFOOTING clause in Section 5.
SUBTITLE	prints a subtitle at the top of each page immediately following the title for all subsequent reports in the current session. See also the SUBTITLE clause in Section 5.
TITLE	prints a title at the top of each page for all subsequent reports in the current session. See also the TITLE clause in Section 5.

AT END STATEMENT

The AT END statement allows you to specify information that is printed at the end of all subsequent reports in the current session unless cancelled or reset by another AT END statement or overridden by an AT END clause. (See the AT END PRINT clause in Section 5.) The syntax of the AT END statement is:

```
AT END [ PRINT print-list [ CENTER ] ] [ ; ]
```

where

print-list

can contain any combination of literals, FORM, SKIP, SPACE, or TAB clauses. *Print-list* can also contain the following elements that can be modified by AS, AS DATE or AS TIME clauses: field names, arithmetic expressions, IF/THEN/ELSE expressions, System Variable clauses, JULIAN-DATE clauses, TIMESTAMP-DATE clauses, TIMESTAMP-TIME clauses, user tables, user variables, or parameter names.

The clauses that can appear as part of a *print-list* are described in Section 5. The other elements are described in Section 3.

Specifying a Field Name in an AT END Statement

If you specify a field name within the *print-list* of an AT END statement, ENFORM prints the same field value as in the last row of the report. A field name appearing within the *print-list* of an AT END statement need not be explicitly included within the following LIST statements. If the field name is not included, ENFORM in effect adds the field name with a NOPRINT clause.

Spacing Considerations

By default the information you specify in the *print-list* of an AT END statement begins printing in the same column position as the leftmost column of the report. Using either the SPACE or TAB clause as the first element in the *print-list* overrides the default. The SPACE or TAB clause can appear anywhere within the *print-list*. For example, the SPACE clause in the following AT END statement causes the two literals to be separated by 15 spaces:

```
AT END PRINT "Report" SPACE 15 "Total Sales";

Report               Total Sales
```

If you specify either a SKIP clause or the symbol / (slash) within a *print-list*, ENFORM advances one or more lines before printing the rest of the AT END *print-list*. The number of lines advanced can be affected by one or more of the following: the digit (if any) following the keyword SKIP, the number of slashes specified, or the option variable @@VSPACE. In the following example, the SKIP clause in the AT END statement causes ENFORM to print two lines:

```
AT END PRINT "End of Report for" SKIP "Region " regnum;

End of Report for
Region 1
```

Using a FORM clause within an AT END statement causes ENFORM to print the remainder of the AT END *print-list* on a new page and to increment the page number.

Using the CENTER clause following the *print-list* of an AT END statement centers the information within the leftmost and rightmost columns of the report.

The CENTER, Option Variable, SKIP, SPACE, and TAB clauses are described in Section 5.

AT END Information for Current Report or All Reports

An AT END statement prints information at the end of all subsequent reports in the current session. The current AT END statement can be reset by specifying a new AT END statement with a different *print-list*. Using an AT END PRINT clause temporarily overrides an AT END statement. An AT END PRINT clause only prints information for the current report.

Cancelling Session-Wide AT END Information

Cancel the AT END statement by specifying the AT END statement without the *print-list* parameter.

AT START STATEMENT

The AT START statement allows you to specify information that is printed just before the first set of column headings for all subsequent reports in the current session unless cancelled or reset by another AT START statement or overridden by an AT START clause. (See the AT START PRINT clause in Section 5.) The syntax of the AT START statement is:

```
AT START [ PRINT print-list [ CENTER ] ] [ ; ]
```

where

print-list

can contain any combination of literals, FORM, SKIP, SPACE, or TAB clauses. *Print-list* can also contain the following elements that can be modified by AS, AS DATE, AS TIME clauses: field names, arithmetic expressions, IF/THEN/ELSE clauses, System Variable clauses, JULIAN-DATE clauses, TIMESTAMP-DATE clauses, TIMESTAMP-TIME clauses, user variables, or parameter names.

Clauses that can be used in a *print-list* are described in Section 5. The other elements are described in Section 3.

If you specify both an AT START statement and a TITLE or SUBTITLE statement, the AT START information is printed after the title or subtitle. The AT START statement differs from the TITLE statement in that the AT START information is printed only on the first page of a report while the title or subtitle is printed on every page of a report.

Specifying a Field Name in an AT START Statement

If you specify a field name within the *print-list* of an AT START statement, ENFORM prints the same field value as in the first row of the report. A field name appearing within the *print-list* of an AT START statement need not be explicitly included within the following LIST statements. If the field name is not included, ENFORM effectively adds it with a NOPRINT clause.

Spacing Considerations

By default the AT START information begins printing in the same column position as the leftmost report column. Using SPACE or TAB clauses as the first element of the *print-list* overrides this default. SPACE or TAB clauses can also appear anywhere within the *print-list.* In the following example, the SPACE clause causes the two literals to be separated by 15 spaces:

```
AT START PRINT "Report" SPACE 15 "Total Sales";

Report                Total Sales
```

If you specify either a SKIP clause or the symbol / (slash) within a *print-list*, the printer advances one or more lines before printing the rest of the AT START *print-list*. The number of lines advanced can be affected by one or more of the following: the digit (if any) following the keyword SKIP, the number of slashes specified, or the option variable @VSPACE. In the following example, the SKIP clause of the AT START statement causes two lines to be printed:

```
AT START PRINT "Report For" SKIP "Region " regnum;

Report For
Region 2
```

Using the FORM clause within an AT START statement causes the remainder of the AT START *print-list* to be printed on a new page and increments the page number.

Using the CENTER clause following the *print-list* of an AT START statement centers the information within the leftmost and rightmost columns of the report.

The CENTER, Option Variable, SKIP, SPACE, and TAB clauses are described in Section 5.

AT START Information for Current Report or All Reports

An AT START statement prints information just before the first set of column headings for all subsequent reports in the current session. The current AT START statement can be reset by specifying a new AT START statement with a different *print-list*. Using the AT START PRINT clause within a LIST statement temporarily overrides the AT START statement for the report generated by the LIST statement.

Cancelling Session-Wide AT START Information

Cancel the AT START statement by using the AT START statement without the *print-list* parameter.

CLOSE STATEMENT

The CLOSE statement allows you to delete a user variable, user aggregate, user table, a parameter, or a record description from the internal table. The syntax of the CLOSE statement is:

```
        ⎧ record-name      ⎫
        ⎪ user-variable-name ⎪
CLOSE   ⎨ user-aggregate-name ⎬ , ... [ ; ]
        ⎪ user-table-name  ⎪
        ⎩ param-name       ⎭
```

where

`record-name`

is the name of a dictionary record description previously accessed by an OPEN statement.

`user-variable-name`

is the name of a user variable previously defined by a DECLARE statement.

`user-aggregate-name`

is the name of a user aggregate previously defined by a DECLARE statement.

`user-table-name`

is the name of a user table previously defined by a DECLARE statement.

`param-name`

name of a parameter previously defined by a PARAM statement.

The Effect of a CLOSE Statement on the Internal Table

As a session progresses, ENFORM maintains an internal table for opened record descriptions, links of record descriptions, and definitions of user variables, user aggregates, user tables, and parameters. This internal table grows with each new OPEN, LINK, DECLARE, or PARAM statement entered.

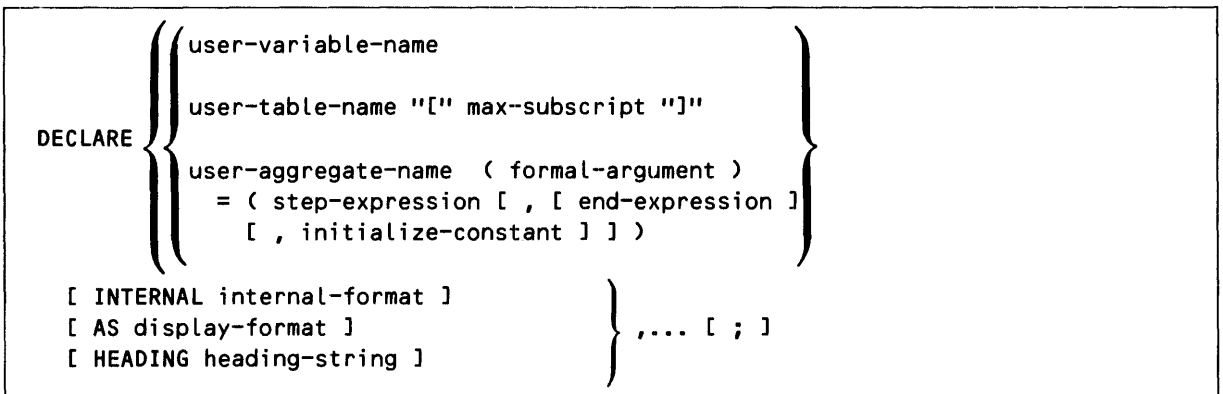
ENFORM has no way of knowing when a table entry is no longer required for a subsequent query within the current session. For this reason, ENFORM allows you to clear unwanted table entries from the internal table with a CLOSE statement. The CLOSE statement does not reclaim space from the internal table, but it does eliminate the effect that unwanted entries might have on subsequent queries. Furthermore, regular use of CLOSE statements reduces the need to qualify fields that are present in more than one record description.

Closing a dictionary record description also clears all links for that particular record description.

An alternative to the CLOSE statement is the DICTIONARY statement or ?DICTIONARY command. Both completely clear the ENFORM internal table and reclaim table space.

DECLARE STATEMENT

The DECLARE statement allows you to define a user variable, user aggregate, or user table. The syntax of the DECLARE statement is:



where

`user-aggregate-name, user-variable-name, or user-table-name`

is the name of the declared element. Names of user-defined elements should conform to the rules described in Section 3.

`" [ " max-subscript " ] "`

is the number of occurrences for the user table; the maximum number allowed is 64. *Max-subscript* must be enclosed within brackets [ ]. Refer to Section 3 for a discussion of subscripts.

`formal-argument`

is the name used to represent the actual argument of the user aggregate.

`step-expression`

is the operation to be performed for each record contributing to the user aggregate.

`end-expression`

is the operation to be performed after all qualifying records for the user aggregate are processed by the *step-expression.*

`initialize-constant`

is the numeric literal that will be the starting value for the user aggregate.

`internal-format`

is the internal format for storing the user variable, aggregate or table.

`display-format`

is the default display format for printing the declared item.

`heading-string`

is a string literal that is the default heading for the declared element. Remember string literals must be enclosed in quotation marks (" ").

Specify the DECLARE statement before you reference a user variable, user aggregate, or user table. ENFORM stores information about each user-defined element in the internal table. This information remains in the internal table until you issue a subsequent CLOSE statement for the user-defined element, issue a DICTIONARY statement or a ?DICTIONARY command, or end the ENFORM session.

Declaring a User Aggregate

User aggregates are processed just like the predefined aggregates. The *step-expression* of a user aggregate can contain:

- Arithmetic expressions
- IF/THEN/ELSE expressions.

The optional *end-expression* is the final operation to be performed after all of the qualifying records are processed by the *step-expression.* The *end-expression* can contain any of the following:

- Predefined aggregates
- Arithmetic expressions
- IF/THEN/ELSE expressions.

By default, the *starting-value* for a user aggregate is zero unless the user aggregate is defined with alphanumeric internal format. In this case, the default value is blanks. An initial value is supplied when you specify *initialize-constant.* If you omit *end-expression* but specify *initialize-constant,* precede *initialize-constant* with two commas.

The following example uses the DECLARE statement to define a user aggregate:

```
DECLARE project (x) = ( IF x < 2000 THEN project + x + .05 * x
  ELSE project + x + .04 * x ) ;
```

More information about user aggregates can be found in Section 3.

Declaring a User Variable or User Table

A user variable or table declaration remains in effect until the end of the current ENFORM session unless you override the user variable or table by declaring a new variable or table with the same name.

By default both user variables and elements in user tables are stored as 64-bit signed integers. To change this default, specify the optional INTERNAL clause described in Section 5. In the following example, the user variable *no-orders* is defined. The INTERNAL clause indicates that *no-orders* is to be stored as alphanumeric with a length of 9 bytes:

```
DECLARE no^orders INTERNAL A9;
```

The default display format for either a user variable or an element in a user table is a fourteen character integer. To change this default, specify the optional AS clause. The display format specified in the AS clause formats the user variable or table element unless you provide an explicit AS clause in the LIST statement. The AS clause is described in Section 5.

Specify the HEADING clause to provide a default heading for either a user variable or a user table. ENFORM uses the default heading for the user variable or table whenever an explicit HEADING clause is not specified in the LIST statement. The HEADING clause is described in Section 5. In the following example, the default display heading "Quarterly Totals" is supplied for the table *qtr^totals*:

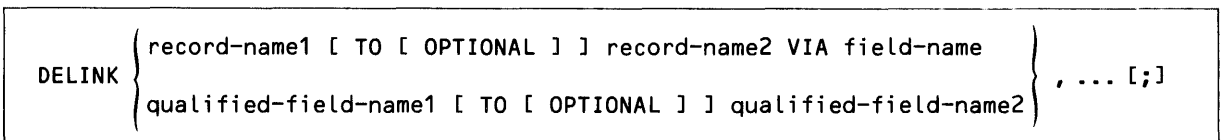
```
DECLARE qtr^totals [4] HEADING "Quarterly Totals";
```

Initialize both user variables and user tables by using the SET statement discussed later in this section.

Section 3 provides more information about user variables and user tables.

DELINK STATEMENT

The DELINK statement allows you to clear a connecting relationship between dictionary record descriptions. The syntax of the DELINK statement is:



where

record-name1 or record-name2

are the names of dictionary record descriptions.

field-name

is the name of a field common to both of the record descriptions.

qualified-field-name1 or qualified-field-name2

are the names of fields uniquely identified as components of *record-name1* and *record-name2* respectively.

Both forms of the DELINK statement are equivalent. The field names must be specified in the DELINK statement in the same order as the corresponding LINK statement. For example, if the LINK statement is:

```
LINK parts TO odetail VIA partnum;
```

the corresponding DELINK statement can be:

```
DELINK parts.partnum TO odetail.partnum;
```

ENFORM stores all links of record descriptions in the current ENFORM internal table. All links apply to all subsequent LIST or FIND statements of the current ENFORM session until you issue a DELINK, CLOSE, DICTIONARY statement, or ?DICTIONARY command. Since unnecessary links can produce undesirable results, delete relationships which do not apply to the current query from the internal table. Use a DELINK statement to clear a a linking relationship between two record descriptions without affecting other links.

If you want to clear all links, use a CLOSE statement, a DICTIONARY statement, or a ?DICTIONARY command. A CLOSE statement for a record description deletes all links referencing that record description from the internal table. A DICTIONARY statement or ?DICTIONARY command clears the entire internal table.

DICTIONARY STATEMENT

The DICTIONARY statement allows you to identify the subvolume containing your dictionary. It also allows you to clear the internal table. The DICTIONARY statement has the same effect as the ?DICTIONARY command. The syntax of the DICTIONARY statement is:

```
DICTIONARY [ dict-subvol-name ] [ ; ]
```

where

dict-subvol-name

is the name of the subvolume where your dictionary files reside. Refer to the *Guardian Operating System Programming Manual* for information on specifying Tandem file names.

The dictionary identified in a DICTIONARY statement must be created by the Data Definition Language compiler. Release T9102C10 of ENFORM accepts only dictionaries produced by DDL version T9100D00 or later. ENFORM issues an error message if an attempt is made to use a dictionary compiled with an earlier version of DDL. These dictionaries must be recompiled with DDL version T9100D00 or later. Refer to the *Data Definition Language (DDL) Programming Manual* for more information about creating and compiling a dictionary.

Identifying the Location of the Dictionary

The location of the dictionary can be indicated by:

- Specifying the volume and subvolume where the dictionary resides as a part of the Command Interpreter ENFORM command. If you do not specify a volume and subvolume, ENFORM assumes the dictionary resides on your default volume and subvolume.
- Specifying either the DICTIONARY statement or ?DICTIONARY command to identify where the dictionary resides. Use of either the DICTIONARY statement or the ?DICTIONARY command overrides the dictionary identified at the time of the Command Interpreter ENFORM command. When a new dictionary is specified, the internal table associated with the old dictionary is cleared.

In the following example, the DICTIONARY statement is used to identify the volume that the dictionary resides on as $data and the subvolume as database:

```
DICTIONARY $data.database;
```

Clearing the Internal Table

Entering the DICTIONARY statement without a volume and subvolume name is a simple means of clearing the entire internal table and reclaiming table space without changing the dictionary. To clear only certain elements of the internal table, refer to the CLOSE and DELINK statements in this section. The elements cleared by the DICTIONARY statement are:

- All dictionary record descriptions from previous OPEN statements.
- All previous links.
- All user variable, user aggregate, and user table definitions.
- All parameter definitions.

EXIT STATEMENT

The EXIT statement terminates the current ENFORM session. The syntax of the EXIT statement is:

```
EXIT [ ; ]
```

The EXIT statement returns control to the invoking process, usually the Command Interpreter. The EXIT statement is the same as the ?EXIT command. Pressing the CTRL and Y terminal keys simultaneously is an alternate way to exit ENFORM.

FIND STATEMENT

The FIND statement allows you to specify the input fields and records that contribute to the target-record and either write output records to a physical file or transmit output records to a host language program. The FIND statement must end with a semicolon. The syntax of the FIND statement is:

```
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 ] ;
```

where

UNIQUE

is an option that prevents duplicate output records. *UNIQUE* adds processing overhead and should be avoided unless you know unwanted duplicate records exist.

output-record-name

is the name of the dictionary record description of the output record.

output-field-name

is the name of a field in the dictionary record description of the output record. ENFORM allows you to qualify *output-field-name*. If you do not qualify *output-field-name*, ENFORM qualifies *output-field-name* with *output-record-name*. In either case, *output-field-name* must be sufficiently qualified to avoid ambiguity between it and any other name specified in the query.

by-item

is an input field name. An input field name must be sufficiently qualified to avoid ambiguity between it and any other name specified in the query.

target-item

is an input field. Valid values for an input field are: a field name, a string literal enclosed in parentheses, a predefined aggregate, a user aggregate, an arithmetic expression, an IF/THEN/ELSE expression, a user table name, a user variable, or a System Variable. An input field name must be sufficiently qualified to avoid ambiguity between it and another name specified in the query.

logical-expression

is an expression returning a true or false value.

→

Output Record Dictionary Description

Before the FIND statement executes, you must include a description of the output record in the dictionary. When the FIND statement executes, ENFORM either writes the output records to the physical file specified in the dictionary or transmits the output records to a host language program.

The following example shows the Data Definition Language (DDL) source code used to produce a dictionary record description for the output record *findfil*.

```
RECORD findfil.
02 custname          PIC X(18).
02 custnum           PIC 9(4).
02 partcost          PIC 9(6)V99.
end
```

ENFORM allows you to specify data base tables (see the description of subscripts in Section 3 for information about data base tables) in the record description of the output record. The following record description contains a data base table named *child*:

```
RECORD newemp.
02 name              PIC X(18).
02 child             PIC X(18)    OCCURS 4 times.
end
```

You can specify that the output records are to be written to a particular physical file either by including the DDL FILE IS clause in the dictionary record description or by specifying the ?ASSIGN command. You can also use the ?ASSIGN command to tell ENFORM to write the records to a file different from the one described in the dictionary. ENFORM writes a physical file generated by the FIND statement with an unstructured file type regardless of the file type you specify in the data dictionary.

In the record description, an output field can differ in data format (picture size, scale, BINARY vs. ASCII numeric string, ...) from the description of the input field as long as the output field has the same data type (numeric or alphanumeric) as the input field. ENFORM performs the data format conversion automatically, including: truncation or padding with blanks for alphanumeric input and output fields of different lengths, binary to ASCII string conversion (or vice versa), and scaling conversion for numeric input and output fields.

Group Definition and Sorting

The BY and BY DESC clauses group and sort records. The appearance of a BY or BY DESC clause in a FIND statement causes every occurrence of a grouped *by-item* value to be written to the output record (unlike the LIST statement where only the first occurrence of a grouped value is written to the output record.) The BY and BY DESC clauses are described in Section 5.

The ASCD and DESC clauses sort records in ascending or descending order. They do not identify a group. The ASCD and DESC clauses are also described in Section 5.

When a FIND statement contains more than one BY, BY DESC, ASCD, or DESC clause, ENFORM determines a major to minor sort precedence by the order in which the BY, BY DESC, ASCD, or DESC clauses appear in the FIND statement. The first clause has highest priority and is sorted first, the next one second priority, down to the last clause.

Output Fields

When the FIND statement executes, ENFORM takes the values from the input elements and stores them in the output field. Values are stored in the output field by name (when the output field has the same name as an input field) or by assignment syntax (:=). ENFORM allows you to qualify *output-field-name*.

ENFORM allows you to use subscripted output field names in a FIND statement when the record description of the output record contains a data base table; however, a subscript range is not valid anywhere in a FIND statement. (Refer to the discussion of subscripts in Section 3 for the syntax used to include subscripts.) For example, the following FIND statement is valid:

```
FIND new emp (name          := emp.name,
              child [ 1 ]   := emp.child1,
              child [ 2 ]   := emp.child2,
              child [ 3 ]   := " ",
              child [ 4 ]    := " "        );
```

The following rules apply to *output-field-name*:

1. *Output-field-name* can be qualified or subscripted.

2. *Output-field-name* cannot contain a subscript range.

3. If *output-field-name* is omitted and the input field name is subscripted or qualified, ENFORM assumes that the same qualifications and subscripts apply to *output-field-name*.

4. *Output-field-name* can be a group name; however, ENFORM might not store the fields within the group in the manner you expect. To understand the way that ENFORM stores group elements, you must first understand what a group is. A group is defined in DDL as any field whose level number (03, 04, ...) is less than that of the next field in the record. Consider, for example, the following DDL record description:

```
RECORD findfl.
FILE IS $mkt.sample.findfl key-sequenced.
02 account-num.
  05 type         PIC 9(4).
  05 num          PIC BINARY 16.
02 custnum        PIC 9(4).
```

Within this record, *account-num* is a group. The data type of a group is always alphanumeric. When a group name is specified as *output-field-name* within a FIND statement, ENFORM stores each element within the group as alphanumeric data. If one of the fields within the group is defined as binary, using a group name as *output-field-name* results in an output record that might contain undesirable data.

Input Elements

The output fields receive values from any combination of the following input elements:

- The value of a field from the input record. The assignment to an *output-field-name* is optional. When you do not specify the *output-field-name*, ENFORM assumes it is the same as the field name from the input record. When the *output-field-name* is omitted, the input field name must be fully qualified.

```
FIND ...
 ( parts.partnum,
    ...) ;
```

- The value of a numeric literal. Remember to enclose numeric literals within parentheses.

```
FIND ...
 (region := (5),
     ... );
```

- The value of a string literal. A string literal must be enclosed in parentheses.

```
FIND ...
  ( jobtitle := ("manager"),
  ... ) ;
```

- The value of an arithmetic expression. Enclose an arithmetic expression within parentheses. Use any combination of numeric fields, numeric literals, numeric user variables, predefined aggregates, or user aggregates for the arithmetic expression.

```
FIND ...
  ( sales := (price * quantity),
  ... ) ;
```

- The value of an IF/THEN/ELSE expression. Enclose the IF/THEN/ELSE expression within parentheses.

```
FIND ...
  (stock := (IF inventory GT 100 THEN inventory
    ELSE ZERO),
  ...);
```

- The value of a user variable. Define the user variable by a DECLARE statement before referencing it in a FIND statement. The SET statement should be used to initialize the user variable. If the user variable is not initialized, the value is zero.

```
DECLARE region;
SET region TO 5;
FIND ...
  ( reg := region,
  ... ) ;
```

- The value of either a user aggregate or a predefined aggregate.

```
FIND ...
  ( BY employee.job,
    employee.salary,
    rate := AVG (salary OVER job),
    ... ) ;
```

- A group name. ENFORM allows you to specify a group name as an input element. Avoid this specification, however, unless the data type of the fields receiving the data is exactly the same as the data type of the input group.

Although ENFORM allows you to specify an output field name as an input element, this specification is not recommended and might lead to unexpected results.

The following rules apply to input elements:

1. An input element can be qualified or subscripted.
2. An input element cannot contain a subscript range.
3. An input element must be sufficiently qualified to avoid any ambiguity between it and any other element specified in the query.

Request-Qualification

Use the WHERE clause to limit the records written to the physical output file or transmitted to the host language program. The WHERE clause is described in Section 5.

Summary Records

A FIND statement can be specified that either creates a new file containing only summary records or transmits only summary records to a host language program. Such summary records contain only the first target-record from each group (created by a BY or BY DESC clause) down to some level. Summary records can only be generated by a query that contains an aggregate.

The two methods of obtaining summary records are:

- Explicitly request summary records by setting the @SUMMARY-ONLY option variable to ON before issuing the FIND statement.
- Implicitly request summary records by specifying only by-items and aggregates over by-items in the query.

When you explicitly request summary records you get target-records summarized down to the lowest level where an aggregate is calculated over that level. For example:

```
SET @SUMMARY-ONLY TO ON;
FIND findfil
 ( BY employee.dept,
   BY employee.job,
   BY employee.empname,
      employee.salary,
  COUNT(employee.empname OVER employee.job);
```

returns one target-record for each job in each department. Only the first employee name (*empname*) for each job is returned.

When you implicitly request summary records, you get target-records summarized down to the lowest level where an aggregate is computed over that level. (A query requesting only by-items and aggregates over ALL is not an implicit request for summary records). For example:

```
SET @SUMMARY-ONLY TO OFF;
FIND findfil
  (BY employee.dept,
  BY employee.job,
  BY employee.empname,
  numemp:= COUNT (employee.empname OVER employee.job));
```

returns one target-record for each job in each department. Only the first employee name (*empname*) for each job is returned.

If you want summary records that consist of only by-items and aggregates to include those where the last value of a by-item has not changed but a subordinate by-item value has changed, then you must include a target-item in the target-list that has not appeared as a by-item or aggregate. Of course, you must also set @SUMMARY-ONLY to OFF. For example:

```
SET @SUMMARY-ONLY TO OFF;
FIND findfil2
     (BY employee.dept,
      BY employee.job,
      BY employee.empname,
        employee.salary,
        numemp: = COUNT (empname OVER job));
```

Statements and Clauses That Do Not Apply to the FIND Statement

ENFORM queries with a FIND statement produce records only. There are no report features such as headings, titles, summary information, and special formatting. The following statements and clauses apply only to queries using the LIST statement and cannot be used with the FIND statement.

AFTER CHANGE clause
AS clause
AS DATE clause
AS TIME clause
AT END statement and AT END PRINT clause
AT START statement and AT START PRINT clause
BEFORE CHANGE clause
CENTER clause
CUM clause
FOOTING statement and clause
FORM clause
HEADING clause
NOHEAD clause
NOPRINT clause
PCT clause
SKIP clause
SPACE clause
SUBFOOTING statement and clause
SUBTITLE statement and clause
SUBTOTAL clause
System Variable clauses
TAB clause
TITLE statement and clause
TOTAL clause

FOOTING STATEMENT

The FOOTING statement allows you to specify a footing to be printed at the bottom of each report page for all reports in the current session unless overridden or reset by another FOOTING statement or temporarily overridden by a FOOTING clause. (See Section 5 for the FOOTING clause.) The syntax of the FOOTING statement is:

```
FOOTING [ print-list [ CENTER ] ] [ ; ]
```

where

print-list

can be any combination of literals, FORM, SKIP, SPACE or TAB clauses. *Print-list* can also contain the following elements that can be modified by AS, AS DATE, or AS TIME clauses: field names, arithmetic expressions, IF/THEN/ELSE expressions, System Variable clauses, JULIAN-DATE clauses, TIMESTAMP-DATE clauses, TIMESTAMP-TIME clauses, user variables, or parameter names.

The clauses that can be part of a *print-list* are discussed in Section 5. The other elements are discussed in Section 3.

Specifying A Field Name Within a FOOTING Statement

If you specify a field name within the *print-list* of a FOOTING statement, ENFORM prints the same field value as in the last row of data on the current page. A field name appearing within the FOOTING statement need not be explicitly included within the following LIST statements. If the field name is not included, ENFORM effectively adds it to the LIST statement with a NOPRINT clause.

Spacing Considerations

By default, the footing begins printing in the same column position as the leftmost report column. Using SPACE or TAB clauses as the first element of the *print-list* overrides this default. SPACE or TAB clauses can also appear anywhere within the *print-list.* In the following example, the SPACE clause causes the two literals to be separated by 15 spaces:

```
FOOTING "Inventory" SPACE 15 "Parts in Stock";
```

The following footing appears at the bottom of the next report that is generated without a LIST statement FOOTING clause.

```
Inventory                Parts in Stock
```

If you specify either a SKIP clause or the symbol slash (/) within a *print-list*, ENFORM advances one or more lines before printing the rest of the FOOTING *print-list*. The number of lines advanced can be affected by one or more of the following: the digit (if any) following the keyword SKIP, the number of slashes specified, or the option variable @VSPACE. In the following example, the SKIP clause in the FOOTING statement causes two lines to be printed:

```
FOOTING "Report 2-A" SKIP "Total Sales";
```

The following footing appears at the bottom of the next report that is generated without a LIST statement FOOTING clause.

```
Report 2-A
Total Sales
```

Using a FORM clause within a FOOTING statement forces a new page. ENFORM continues with the remainder of the FOOTING *print-list*. The page number remains the same. A single logical page can span multiple physical pages such that a TITLE can appear on one page, the data on the next, and a FOOTING on the next. The same page number applies to all physical pages in a logical page.

Using the CENTER clause within the FOOTING statement centers the footing within the leftmost and rightmost columns of the report.

The CENTER, Option Variable, SKIP, SPACE, and TAB clauses are described in Section 5.

Footing for Current Report or All Reports

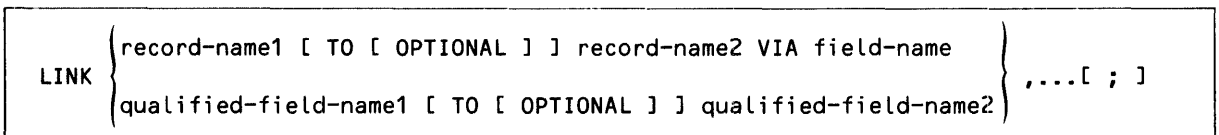
A FOOTING statement prints a footing at the bottom of each page for all subsequent reports in the current session. Specifying a new FOOTING statement with a different *print-list* resets the current FOOTING statement. Using the FOOTING clause within a LIST statement temporarily overrides the FOOTING statement. A FOOTING clause within a LIST statement prints a footing only for the current report.

Cancelling Session-Wide Footing

Cancel the FOOTING statement by using the FOOTING statement without the *print-list* parameter.

LINK STATEMENT

The LINK statement allows you to specify a connecting relationship between dictionary record descriptions. The syntax of the LINK statement is:



where

record-name1 and record-name2

are the names of dictionary record descriptions containing a common field.

field-name

is the name of a field common to both of the record descriptions.

qualified-field-name1 and qualified-field-name2

are the names of fields uniquely identified as components of the record descriptions being linked.

OPTIONAL

indicates that all of the records from the first file specified in the LINK statement are included whether or not records with matching linking fields are found in the second file. All of the records in the first file must have a value in the linking field or a sort operation might fail.

Only one LINK *OPTIONAL* statement can be specified for a LIST or FIND statement. While LINK *OPTIONAL* is in effect, other linking relationships such as another LINK statement or a linking WHERE clause cannot be specified.

A LINK statement does not actually link physical files, but forms a logical relationship between the dictionary record descriptions. Up to 32 linking relationships can exist.

Using the LINK Statement to Connect Files

The LINK statement is a convenient way of expressing a connecting relationship between two record descriptions. In subsequent LIST or FIND statements, ENFORM treats the linked record descriptions as one logical record description. The LINK statement is equivalent to the following WHERE clause in a LIST or FIND statement:

```
WHERE qualified-field-name-1 EQ qualified-field-name-2;
```

Before a LIST or FIND statement is executed, ENFORM actually converts the LINK statement into a WHERE clause and adds it to the LIST or FIND statement. When several LINK statements exist, an extra clause for each LINK statement is added to the WHERE clause in the LIST or FIND statement.

Clearing a LINK

ENFORM stores each LINK and LINK *OPTIONAL* statement in the internal table. The LINK statement remains in effect until:

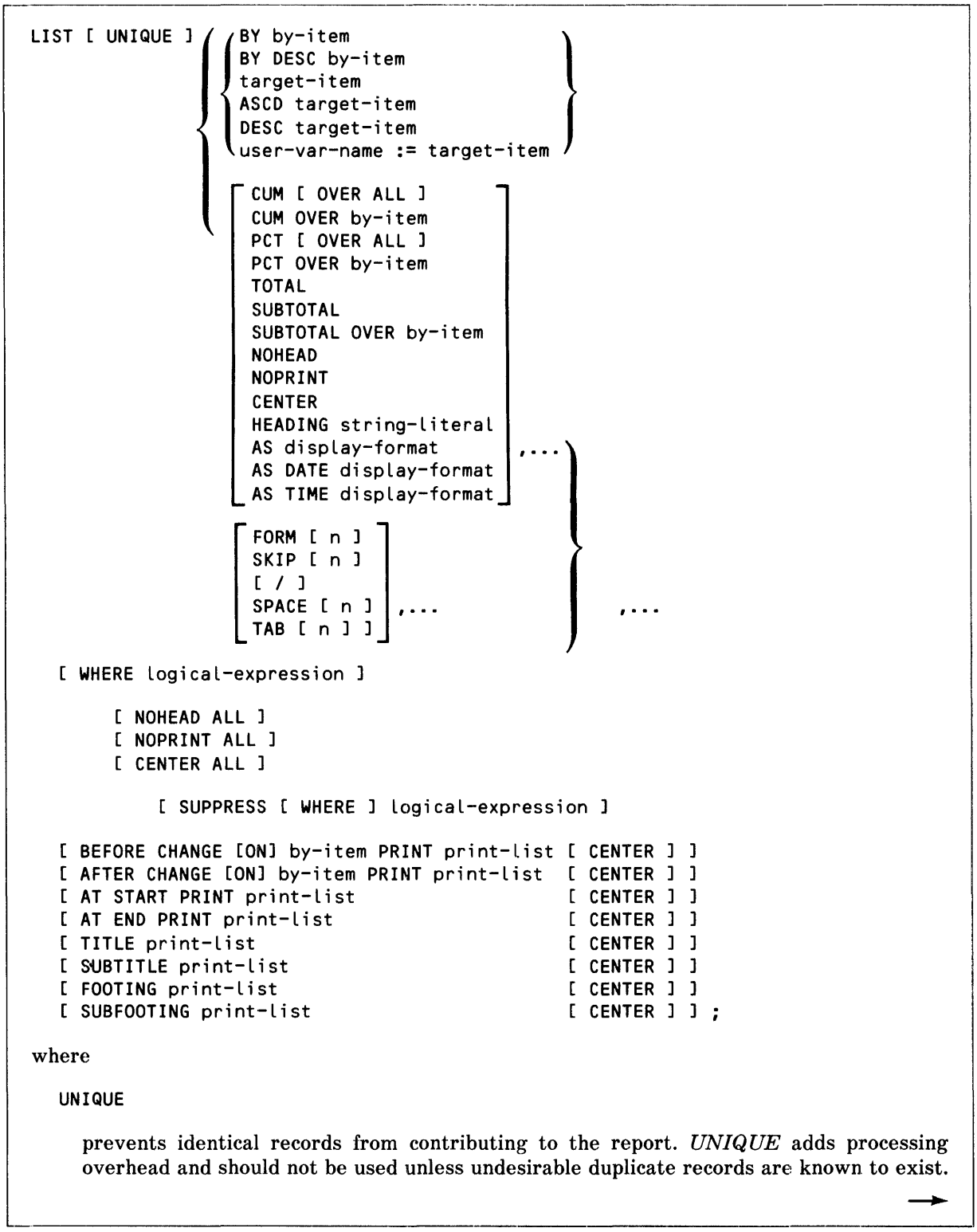
- a CLOSE statement clears one of the record descriptions
- a DELINK statement clears the link
- a DICTIONARY statement or ?DICTIONARY command clears the entire internal table

LINK statements that are not used in processing a query act as unwanted WHERE clauses in subsequent LIST or FIND statements. Thus, any LINK statements that might be unnecessary should be cleared when no longer needed.

Unlike the LINK statement, a WHERE clause associated with a LIST or FIND statement clears after ENFORM processes the statement.

LIST STATEMENT

The LIST statement allows you to select the information printed in a report and prints the report. The LIST statement must end with a semicolon. The syntax of the LIST statement is:



where

UNIQUE

prevents identical records from contributing to the report. *UNIQUE* adds processing overhead and should not be used unless undesirable duplicate records are known to exist.

→

`by-item`

is the name of a field modified by a BY or a BY DESC clause. The report is sorted and grouped according to the value of this field.

`target-item`

is a record name, a field name, a literal, a predefined aggregate, a user aggregate, an arithmetic expression, an IF/THEN/ELSE expression, a user table name, a user variable, or a system variable clause.

`user-var-name`

is the name of a user variable.

`string-literal`

is one or more alphanumeric characters enclosed within quotation marks.

`display-format`

is the format in which you want an element displayed.

`/`

is a symbol that is equivalent to the SKIP clause. You can specify as many of these symbols as you want to indicate the number of lines to advance.

`logical-expression`

is an expression returning a true or false value.

`print-list`

contains any combination of literals, FORM, SKIP, SPACE or TAB clauses. A *print-list* can also contain the following elements that can be modified by AS, AS DATE, or AS TIME clauses: field names, arithmetic expressions, IF/THEN/ELSE expressions, system variable clauses, JULIAN-DATE clauses, TIMESTAMP-DATE clauses, TIMESTAMP-TIME clauses, user variables, and parameter names.

Group Definition and Sorting

BY and BY DESC clauses group and sort records. ENFORM prints the *by-item* value for a group only for the first record of all the records that have the same value for the *by-item*. *By-items* can be referred to by CUM OVER, PCT OVER, SUBTOTAL OVER, AFTER CHANGE, and BEFORE CHANGE clauses. These clauses are described in Section 5.

ASCD and DESC clauses sort records in ascending or descending order. They do not identify a group. The ASCD and DESC clauses are described in Section 5.

When a LIST statement contains more than one BY, BY DESC, ASCD, or DESC clause, ENFORM determines a major to minor sorting precedence. ENFORM determines the sorting precedence by the order in which the clauses appear in the LIST statement. The first clause has highest priority and is sorted first, the next one second priority, down to the last specified clause.

How Values Are Displayed in Report Columns

ENFORM displays *target-items* and *by-items* in report columns, one column per item. If a record name is a *target-item* in a LIST statement, ENFORM expands the record to as many *target-items* as the number of elementary fields in the record. If a field within the record is described with an OCCURS clause, ENFORM prints each occurrence of the field in a separate column of the report. Do not specify a record name as a *target-item* if it is the same as a field name in a record. Refer to Section 3 for information about field names. A *target-item* in a LIST statement can be:

- a record name

```
LIST ...,
  parts;
```

- the name of a field

```
LIST ...,
   partnum;
```

- a numeric literal. Enclose numeric literals within parentheses.

```
LIST ...,
  (5);
```

- a string literal.

```
LIST ...,
  "manager";
```

- an arithmetic expression enclosed in parentheses. The expression can use any combination of numeric fields, numeric literals, numeric user variables, predefined aggregates, user aggregates, JULIAN-DATE clauses, or system variable clauses.

```
LIST ...,
  (price * quantity);
```

- an IF / THEN / ELSE expression. Enclose IF/THEN/ELSE expressions in parentheses.

```
LIST ...,
  (IF inventory GT 0 THEN inventory ELSE ZERO);
```

- a user variable. Define the user variable with a DECLARE statement. The variable should be either initialized with a SET statement or assigned a value with assignment syntax. If the variable value is neither initialized nor assigned, the default value is zero. A user variable cannot be assigned the values of a subscript range.

```
DECLARE new-sal;
SET new-sal TO (salary * 1.20);
LIST ... ,
    new-sal;
```

- a system variable clause.

```
LIST ... ,
  @LINENO AS I5;
```

- a user aggregate or predefined aggregate.

```
LIST ...,
  COUNT (branchnum OVER regnum);
```

- a field or user table with a subscript range.

```
LIST ... ,
  months [ 1:6 ];
```

- a parameter name. Parameters that have not been declared with an alphanumeric internal format must be enclosed in parentheses. Parameters declared with an alphanumeric internal format must not be enclosed within parentheses.

```
PARAM num;
...
LIST ...,
     (num);
```

ENFORM allows you to specify a group name as a target-item within a LIST statement; however, ENFORM might not display the fields within the group in the manner you expect. To understand the way that ENFORM displays group elements, you must first understand what a group is. A group is defined in DDL as any field whose level number (03, 04, 05, ...) is less than that of the next field. For example, consider the following DDL record description:

```
RECORD test.
FILE IS "$mkt.sample.test" key-sequenced.
02 account-num.
  05 type          PIC 9(4).
  05 num           TYPE BINARY 16.
02 custnum         PIC 9(4).
end
```

Within this record, *account-num* is a group. The data type of group is always alphanumeric. When a group name is specified as a target-item within a LIST statement, ENFORM displays each field within the group as alphanumeric data. If one of the fields within a group contains binary data, using a group name as a target-item causes undesirable results. For example, specifying:

```
LIST account-num;
```

causes ENFORM to display *account-num* without first converting it to a readable form.

Request-Qualification

Use the WHERE clause to limit the records that contribute to the report. The WHERE clause is described in Section 5.

Conditional Printing

Use the SUPPRESS clause to define a condition or conditions that prevent specific records from printing throughout a report. ENFORM still includes the suppressed records in AFTER CHANGE and BEGIN CHANGE clauses, subtotals, totals, and other calculations specified for the report. Note that the value of the first record of an AFTER CHANGE clause or the last record of a BEFORE CHANGE clause is used for the *print-list* whether or not that record is printed. The SUPPRESS clause is described in Section 5.

Summary Reports

Summary reports contain only the first target-record from each group (created by a BY or BY DESC clause) down to some level. Summary reports can only be generated by a query that contains an aggregate.

The two methods of obtaining a summary report are:

- Explicitly request a summary report by setting the @SUMMARY-ONLY option variable to ON before issuing the LIST statement.
- Implicitly request a summary report by specifying only *by-items* and aggregates over *by-items* in the query.

When you explicitly request a summary report, you get a report summarized down to the lowest level where an aggregate is calculated over that level. For example:

```
SET @SUMMARY-ONLY TO ON;
LIST BY dept, BY job, BY empname, salary,
  COUNT(empname OVER job);
```

returns one record for each job in each department. Only the first employee name (*empname*) for each job is returned.

When you implicitly request a summary report, you get a report summarized down to the lowest level where an aggregate is computed over that level. (A query requesting only *by-items* and aggregates over ALL is not an implicit request for a summary report). For example:

```
SET @SUMMARY-ONLY TO OFF;
LIST BY dept, BY job, BY empname,
  COUNT (empname OVER job);
```

returns one record for each job in each department. Only the first employee name (*empname*) for each job is returned.

If you want a report that consists of only *by-items* and aggregates to contain all report lines (including those where the last value of a by-item has not changed but a subordinate by-item value has changed), then you must include a *target-item* in the target-list that has not appeared as a *by-item* or aggregate. This *target-item* can be modified by the NOPRINT clause if desired. Of course, you must also set @SUMMARY-ONLY to OFF. For example:

```
SET @SUMMARY-ONLY TO OFF;
LIST BY dept, BY job, BY empname, salary NOPRINT,
  COUNT (empname OVER job);
```

returns more than one record for each job in each department. All the employee names (*empname*) for each job are returned.

Optional Clauses

These clauses are optional and are described in Section 5. If you use one of these clauses to modify a *target-item* that contains a subscript range, the clause modifies each element in the range. Briefly, the clauses do the following:

- AFTER CHANGE clause prints information preceding the records for each *by-item* within a printed report.
- AS clause specifies the display format for a *target-item* or a *by-item*
- AS DATE clause specifies the date format for printing a date
- AS TIME clause specifies the time format for printing a time
- AT END PRINT clause prints information at the end of a report
- AT START PRINT clause prints information at the beginning of a report
- BEFORE CHANGE clause prints information following the records for each *by-item* within a printed report
- CENTER clause centers an object within its context
- CUM clause prints a running total for a *target-item* or *by-item*
- FOOTING clause prints a footing for a report
- HEADING clause specifies a column heading for a report
- NOHEAD clause suppresses the printing of the column heading for a *target-item* or *by-item*
- NOPRINT clause suppresses the printing of a *target-item* or *by-item*
- PCT clause prints a percentage for a *target-item* or *by-item*
- SUBFOOTING clause prints a subfooting for a report
- SUPPRESS clause defines a condition that prevents specific records from printing in a report.
- SUBTOTAL clause prints a subtotal for a *target-item* within each *by-item*
- SUBTITLE clause prints a subtitle for a report
- TITLE clause prints a title for a report
- TOTAL clause prints a total for a *target-item* or *by-item*
- WHERE clause limits the records that contribute to the report.

OPEN STATEMENT

The OPEN statement accesses the dictionary record description. The syntax of the OPEN statement is:

```
OPEN { record-name                                          }
     { record-name2 [ AS ] COPY [ OF ] record-name1 }  , ... [ ; ]
```

where

record-name, record-name1, or record-name2

are the names of dictionary record descriptions.

Specifying an OPEN statement does not actually open a physical file, rather the OPEN statement causes ENFORM to access the dictionary record description of the file. One OPEN statement can open several record descriptions. ENFORM stores the record description in the internal table. The record description remains open until:

- A CLOSE statement clears the record description.
- Either a DICTIONARY statement or ?DICTIONARY command clears the entire internal table.
- The current ENFORM session ends.

Using OPEN AS COPY OF

The OPEN AS COPY OF statement is useful when a record description is designed so that records within a file relate to other records within the identical file. Because LINK statements only associate record descriptions with different record names, the OPEN AS COPY OF statement refers to the same record description by a different record name.

Note that although the OPEN AS COPY OF statement opens *record-name2*, you must open *record-name1* before specifying this statement.

PARAM STATEMENT

The PARAM statement allows you to name and define a parameter that can receive a value from a Command Interpreter PARAM command. The syntax of the PARAM statement is:

```
PARAM { param-name [ INTERNAL internal-format ] } , ... [ ; ]
```

where

param-name

is the name of the parameter being defined. The name must conform to the naming conventions described in Section 3.

internal-format

is the internal format for storing the parameter. Valid values for *internal-format* are:

An	alphanumeric, where *n* is the length.
In	integer, where *n* is the length.
Fw.d	fixed, where *w* is the number of digits and *d* is the number of decimal places.

The default is a 64-bit signed integer.

Up to 32 parameters can be defined per ENFORM session. The SET statement can initialize the value of the parameter. The following PARAM statement defines a parameter named *regno*:

```
PARAM regno INTERNAL I2;
```

Values for parameters can only be passed to an executing compiled query file. (A compiled query file is created by the ?COMPILE command described in Section 6.) Before executing the stored compiled query file, a Command Interpreter PARAM command can be used to pass a value for the parameter. The value specified by the Command Interpreter PARAM command supersedes the parameter value specified by a SET statement. The Command Interpreter PARAM command must precede the Command Interpreter ENFORM command. Refer to the *GUARDIAN Operating System Command Language and Utilities Manual* for more information about the Command Interpreter PARAM command.

Parameter values can also be passed through the host language interface. Refer to the *ENFORM User's Guide* for information about the host language interface.

How ENFORM Treats Parameters

ENFORM treats a parameter syntactically as if it were a literal. ENFORM handles parameters declared with an alphanumeric internal format as string literals. ENFORM handles all other parameters as numeric literals. When you specify a parameter with a non-alphanumeric internal format as a target-item or as an item in a print-list, you must enclose the parameter in parentheses just as you would an actual numeric literal. For example, suppose *rept*, a compiled query file, contains the following ENFORM statements:

```
PARAM reptnum I3;
TITLE "REPORT ", (reptnum);
OPEN parts;
LIST parts;
```

Notice that *reptnum* is enclosed in parentheses when specified in the TITLE statement. To provide a value for *reptnum*, enter the following Command Interpreter PARAM command:

```
:PARAM reptnum 333
```

If you then specify the ENFORM command:

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

the resulting report is:

```
REPORT 333

 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
  ...   ...                       ...  ...       ...
```

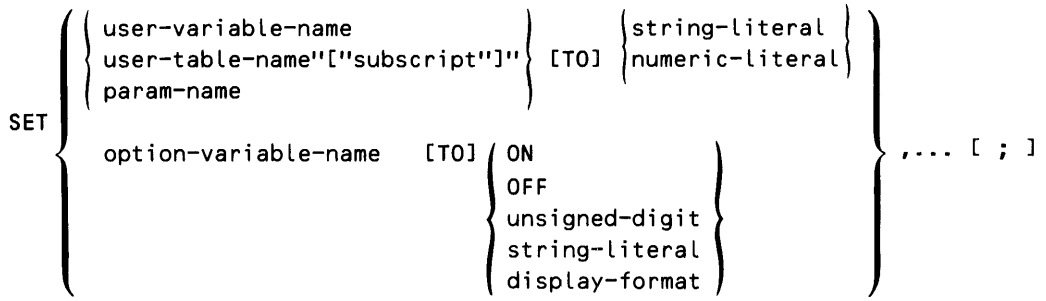
You can use a parameter wherever a literal is allowed. In certain cases, ENFORM allows you to use a parameter but treats the parameter strictly as a numeric literal. Therefore, you cannot change the value of the parameter at execution time. ENFORM treats a parameter strictly as a numeric literal when you use the parameter as:

- a subscript
- the "max-subscript" in the declaration of a user table
- the integer in a FORM, SKIP, TAB, or SPACE clause
- the integer in a pattern-match string of a logical expression.

ENFORM issues a warning message when you have specified a parameter in this manner.

SET STATEMENT

The SET statement allows you to initialize or reset a user variable, user table, or a parameter. The SET statement also allows you to reset option variables. The syntax of the SET statement is:



where

`user-variable-name or user-table-name"["subscript"]"`

is the name of an element defined by a DECLARE statement.

`param-name`

is the name of a parameter defined by a PARAM statement.

`string-literal`

is one or more alphanumeric characters enclosed within quotation marks.

`unsigned-digit`

is an unsigned numeric literal.

`option-variable-name`

is the name of an option variable.

`numeric-literal`

is an integer.

`display-format`

is the default display format enclosed in quotation marks for printing dates or times when AS DATE * and AS TIME * clauses are used.

Initializing User Defined Elements

When initializing a user variable, user table, or a parameter, specify values that are consistent with the internal format type. If the internal format type is alphanumeric, specify a a string literal; if the internal format type is numeric, specify a numeric literal. In the following example, the user variable u-var is defined as numeric by the DECLARE statement and set to the value of 99 by the SET statement:

```
DECLARE u-var INTERNAL I2;
SET u-var TO 99;
```

The SET statement can initialize a user table. An unsigned numeric integer subscript must be used. In the following example, the third element of the months table is set to MARCH:

```
DECLARE months [12] INTERNAL A10;
SET months [3] TO "MARCH";
```

Redefining Option Variables

Use the SET statement to redefine option variables. There are several option variables, each with default values assigned by ENFORM. Refer to the description of the Option Variable Clause in Section 5 for information about valid values for the option variables.

SUBFOOTING STATEMENT

The SUBFOOTING statement allows you to specify a subfooting to be printed at the bottom of each report page for all reports in the current session unless overridden or reset by another SUBFOOTING statement or temporarily overridden by a SUBFOOTING clause. See the SUBFOOTING clause in Section 5. The syntax of the SUBFOOTING statement is:

```
SUBFOOTING [ print-list [ CENTER ] ] [ ; ]
```

where

print-list

can be any combination of literals, FORM, SKIP, SPACE or TAB clauses. A *print-list* can contain the following elements that can be modified by AS, AS DATE or AS TIME clauses: field names, arithmetic expressions, IF/THEN/ELSE expressions, System Variable clauses, JULIAN-DATE clauses, TIMESTAMP-DATE clauses, TIMESTAMP-TIME clauses, user variables, or parameter names.

The clauses that can be used in a *print-list* are discussed in Section 5. The other elements are discussed in Section 3.

Specifying Field Names Within a SUBFOOTING Statement

If you specify a field name within a *print-list* of a SUBFOOTING statement, the field value printed is the same value as in the last row of data on the current page. A field appearing within the SUBFOOTING statement need not be explicitly included within the following LIST statements. If a SUBFOOTING statement field is not included, ENFORM effectively adds it to the LIST statement with a NOPRINT clause.

Spacing Considerations

By default, the subfooting begins in the same column position as the leftmost report column. Using either SPACE or TAB clauses as the first element of the *print-list* overrides this default. The SPACE or TAB clauses can also appear anywhere within the *print-list*. In the following example, the SPACE clause causes the two literals to be separated by 15 spaces:

```
SUBFOOTING "Inventory" SPACE 15 "Parts in Stock";
```

The following subfooting will appear at the bottom of the next report that is generated without a LIST statement SUBFOOTING clause.

```
Inventory               Parts in Stock
```

If you specify either a SKIP clause or the symbol / (slash) within a *print-list*, the printer advances one or more lines before printing the rest of the SUBFOOTING *print-list*. The number of lines advanced can be affected by one or more of the following: the digit (if any) following the keyword SKIP, the number of slashes specified, or the option variable @VSPACE. In the following example, the SKIP clause of the SUBFOOTING statement causes the subfooting to print on two lines:

```
SUBFOOTING "Report 2-A" SKIP "Total Sales";
```

The following subfooting will appear at the bottom of the next report that is generated without a LIST statement SUBFOOTING clause.

```
Report 2-A
Total Sales
```

Using the FORM clause within a SUBFOOTING statement forces a new page. The remainder of the SUBFOOTING *print-list* is printed on the new page. The page number remains the same. A single logical page can span multiple physical pages such that a TITLE can appear on one page, the data on the next, and a SUBFOOTING on the next. The same page number applies to all physical pages in a logical page.

Using the CENTER clause within the SUBFOOTING statement centers the subfooting within the leftmost and rightmost columns of the report.

The CENTER, Option Variable, SKIP, SPACE, and TAB clauses are described in Section 5.

Subfooting for Current Report or All Reports

A SUBFOOTING statement prints a subfooting at the bottom of each page for all subsequent reports in the current session. Specifying a new SUBFOOTING statement with a new *print-list* resets the current SUBFOOTING statement. A SUBFOOTING statement can be temporarily overridden by a SUBFOOTING clause within a LIST statement. A SUBFOOTING clause only prints a subfooting for the current report.

Cancelling Session-Wide Subfooting

A SUBFOOTING statement can be cancelled by using the SUBFOOTING statement without the *print-list* parameter.

SUBTITLE STATEMENT

The SUBTITLE statement allows you to specify a subtitle for all subsequent reports in the current session. The subtitle is printed at the top of each page immediately following the title. The SUBTITLE statement can be overridden or reset by another SUBTITLE statement or temporarily overridden by a SUBTITLE clause. The syntax of the SUBTITLE statement is:

```
SUBTITLE [ print-list [ CENTER ] ] [ ; ]
```

where

print-list

can contain any combination of literals, FORM, SKIP, SPACE, or TAB clauses. A *print-list* can also contain the following elements that can be modified by AS, AS DATE, or AS TIME clauses: field names, arithmetic expressions, IF/THEN/ELSE expressions, System Variable clauses, JULIAN-DATE clauses, TIMESTAMP-DATE clauses, TIMESTAMP-TIME clauses, user variables, or parameter names.

The clauses that can be specified in a *print-list* are discussed in Section 5. The other elements are discussed in Section 3.

Specifying a Field Name Within a SUBTITLE Statement

If you specify a field name within the *print-list* of a SUBTITLE statement, the field name has the same value as in the first row of the page. A field name appearing within the SUBTITLE statement need not be explicitly included within the following LIST statements. If the field name is not included, ENFORM effectively adds it with a NOPRINT clause.

Spacing Considerations

By default the subtitle begins printing in the same column position as the leftmost report column. Using SPACE or TAB clauses as the first element of the *print-list* overrides this default. SPACE or TAB clauses can also appear anywhere within the *print-list*. In the following example, the SPACE clause causes the two literals to be separated by 15 spaces:

```
SUBTITLE "Inventory" SPACE 15 "Parts in Stock";

Inventory               Parts in Stock
```

If you specify either a SKIP clause or the slash symbol (/) within a *print-list*, ENFORM advances one or more lines before printing the rest of the SUBTITLE *print-list*. The number of lines advanced can be affected by one or more of the following: the digit (if any) following the keyword SKIP, the number of slashes specified, or the option variable @VSPACE. In the following example, the SKIP clause of the SUBTITLE statement causes two lines to be printed:

```
SUBTITLE "Report 2-A" SKIP "Total Sales";

Report 2-A
Total Sales
```

Using the FORM clause within a SUBTITLE statement forces a new page. ENFORM prints the remainder of the SUBTITLE *print-list*, starting at the top of the next physical page. The page number remains the same. A single logical page can span multiple physical pages such that a SUBTITLE can appear on one page, the data on the next, and a FOOTING on the next. The same page number applies to all physical pages in a logical page.

Using the CENTER clause within a SUBTITLE statement centers the subtitle within the leftmost and rightmost columns of the report.

The CENTER, Option Variable, SKIP, SPACE, and TAB clauses are described in Section 5.

Subtitle for Current Report or All Reports

A SUBTITLE statement prints a subtitle at the top of each page immediately following the title for all subsequent reports in the current session. The SUBTITLE statement can be reset by specifying a new SUBTITLE statement with a different *print-list*. A SUBTITLE clause within a LIST statement temporarily overrides the SUBTITLE statement. A SUBTITLE clause within a LIST statement prints a subtitle only for the current report.

Cancelling Session-Wide Subtitle

Cancel a SUBTITLE statement by using the SUBTITLE statement without the *print-list* parameter.

TITLE STATEMENT

The TITLE statement allows you to specify a title to be printed the top of each page for all subsequent reports in the current session unless cancelled or reset by another TITLE statement or temporarily overridden by a TITLE clause. (See the TITLE clause in Section 5.) The syntax of the TITLE statement is:

```
TITLE [ print-list [ CENTER ] ] [ ; ]
```

where

print-list

can contain any combination of literals, FORM, SKIP, SPACE, or TAB clauses. A *print-list* can also contain the following elements that can be modified by AS, AS DATE, or AS TIME clauses: field names, arithmetic expressions, IF/THEN/ELSE expressions, System Variable clauses, JULIAN-DATE clauses, TIMESTAMP-DATE clauses, TIMESTAMP-TIME clauses, user variables, or parameter names.

The clauses that can be used in a *print-list* are discussed in Section 5. The other elements are discussed in Section 3.

Specifying a Field Name Within a TITLE Statement

If you specify a field name within the *print-list* of a TITLE statement, the field has the same value as in the first row of the page. A field appearing within the TITLE statement need not be explicitly included within the following LIST statements. If it is not included, ENFORM effectively adds it with a NOPRINT clause.

Spacing Considerations

By default the title begins printing in the same column position as the leftmost report column. Using SPACE or TAB clauses as the first element of the *print-list* overrides this default. SPACE or TAB clauses can also appear anywhere within the *print-list*. In the following example, the SPACE clause causes the two literals to be separated by 15 spaces:

```
TITLE "Inventory" SPACE 15 "Parts in Stock";

Inventory               Parts in Stock
```

If you specify either a SKIP clause or the symbol / (slash) within a *print-list*, the printer advances one or more lines before printing the rest of the TITLE *print-list*. The number of lines advanced can be affected by one or more of the following: the digit (if any) following the keyword SKIP, the number of slashes specified, or the option variable @VSPACE. In the following example, the SKIP clause of the TITLE statement causes two lines to be printed:

```
TITLE "Report 2-A" SKIP "Total Sales";

Report 2-A
Total Sales
```

Using the FORM clause within a TITLE statement forces a new page. The remainder of the TITLE *print-list* is printed, starting at the top of the next physical page. The page number remains the same. A single logical page can span multiple physical pages such that a TITLE can appear on one page, the data on the next, and a FOOTING on the next. The same page number applies to all physical pages in a logical page.

Using the CENTER clause within a TITLE statement centers the title within the leftmost and rightmost columns of the report.

The CENTER, Option Variable, SKIP, SPACE, and TAB clauses are described in Section 5.

Title for Current Report or All Reports

A TITLE statement prints a title at the top of each page for all subsequent reports in the current session. Specifying a new TITLE statement resets the TITLE statement to the value of the new *print-list*. A TITLE statement can be temporarily overridden by a TITLE clause within a LIST statement. A TITLE clause within a LIST statement only prints a title for the current report.

Cancelling Session-Wide Title

A TITLE statement can be cancelled by using the TITLE statement without the *print-list* parameter.

SECTION 5

CLAUSES

This section describes the syntax of the ENFORM clauses. The clauses are specified in alphabetical order.

ENFORM clauses are components of statements. They provide additional specifications of the ENFORM program to be performed. Most of the ENFORM clauses remain in effect only for the query associated with the LIST OR FIND statement of which they are a part.

An ENFORM program with a FIND statement produces records only; it has no report features such as headings, titles, summary information, and special formatting. The following clauses are applicable only to ENFORM programs with a LIST statement and cannot be used in programs with a FIND statement:

- AFTER CHANGE clause
- AS clause
- AS DATE clause
- AS TIME clause
- AT END PRINT clause
- AT START PRINT clause
- BEFORE CHANGE clause
- CENTER clause
- CUM clause
- FORM clause
- FOOTING clause
- HEADING clause
- NOHEAD clause
- Several of the Option Variable clauses
- NOPRINT clause
- PCT clause
- SKIP clause
- SPACE clause
- SUBFOOTING clause
- SUBTITLE clause
- SUBTOTAL clause
- SUPPRESS clause
- System Variable clauses
- TAB clause
- TITLE clause
- TOTAL clause
- WHERE clause

Table 5-1 shows the ENFORM clauses and their functions.

Table 5-1. ENFORM Clauses and Their Functions

Field Selection, Grouping, and Sorting	
ASCD and DESC	sort target-records in ascending or descending order respectively according to the value of a specified field.
BY and BY DESC	group and sort target-records according to the value of a specified field.
SUPPRESS	eliminates certain records from being printed in the report, but does not limit the records from contributing to the report calculations.
WHERE	selects which records will contribute to the output.
Calculating Running Total, Total, Subtotal, and Percentage	
CUM	prints a running total for a numeric target-item. The CUM OVER clause prints the running group total for a numeric target-item.
PCT	prints the value of the percentage of the grand total for a numeric target-item. The PCT OVER clause prints the percentage that each grouped value is of the total of all values in the group.
SUBTOTAL	prints the value of the target-item subtotals for each group.
TOTAL	prints the value of the grand total for a target-item.
Extracting Current Values for Date, Time, Line Number, and Page	
System Variable	return the current value for the current date, time, line number, and page number.
TIMESTAMP-DATE	extracts the date portion of a timestamp field that has been created by the GUARDIAN procedure TIMESTAMP.
TIMESTAMP-TIME	extracts the time portion of a timestamp field that has been created by the GUARDIAN procedure TIMESTAMP.
Printing User Supplied Information on a Report	
AFTER CHANGE	prints information for the current report preceding the records for each grouped field value.
AT END PRINT	prints information at the end of the current report. See also the AT END statement in Section 4.
AT START PRINT	prints information just before the first set of column headings for the current report. See also the AT START statement in Section 4.
BEFORE CHANGE	prints information for the current report following the records for each grouped field value.

Table 5-1. ENFORM Clauses and Their Functions (Concluded)

FOOTING	prints a footing at the bottom of each page for the current report. See also the FOOTING statement in Section 4.
SUBFOOTING	prints a subfooting at the bottom of each page immediately preceding the footing for the current report. See also the SUBFOOTING statement in Section 4.
SUBTITLE	prints a subtitle at the top of each page immediately following the title for current report. See also the SUBTITLE statement in Section 4.
TITLE	prints a title at the top of each page for the current report. See also the TITLE statement in Section 4.
Converting Data to Internal or Display Format	
AS	specifies a display format for printing a target-item or by-item.
AS DATE	specifies the display format for printing a date.
AS TIME	specifies the display format for printing a time.
INTERNAL	specifies the storage format for a user defined element.
JULIAN-DATE	translates date information to internal format.
Formatting a Report	
CENTER	centers a target-item within its context.
FORM	controls when to skip to a new page.
HEADING	overrides the default column title for a target-item or by-item in a report.
NOHEAD	suppresses the printing of the column heading of a target-item or by-item.
NOPRINT	suppresses the printing of a target-item or by-item and its column heading.
SKIP	specifies how many lines to skip.
SPACE	specifies horizontal spacing.
TAB	specifies in which report column a target-item or by-item is to begin printing.
Supplying Operational Variables	
Option Variable	redefines the default values for several operational variables..

AFTER CHANGE CLAUSE

The AFTER CHANGE clause prints information preceding the records for each group for the current report. The AFTER CHANGE clause is an optional part of a LIST statement. The syntax of the AFTER CHANGE clause is:

```
AFTER CHANGE [ ON ] by-item PRINT print-list [ CENTER ]
```

where

by-item

is the name of a field that has been grouped by a BY or BY DESC clause.

print-list

contains any combination of literals, FORM, SKIP, SPACE or TAB clauses. A *print-list* can also contain the following elements that can be modified by AS, AS DATE, or AS TIME clauses: field names, arithmetic expressions, IF/THEN/ELSE expressions, System Variable clauses, JULIAN-DATE clauses, TIMESTAMP-DATE clauses, TIMESTAMP-TIME clauses, user variables, or parameter names.

The clauses that can be used in a *print-list* are described in this section. The other elements are described in Section 3.

The AFTER keyword of the AFTER CHANGE clause refers to the values printed, not to the location of the printed information. For the exact location of where the AFTER CHANGE information is printed within a report, refer to the *ENFORM Users Guide.*

When more than one AFTER CHANGE clause is specified, the specified information prints in the order in which the AFTER CHANGE clauses are entered.

Specifying a Field Name Within an AFTER CHANGE Clause

If you specify a field name within the *print-list* of an AFTER CHANGE clause, ENFORM uses the same field value as in the first row of the next group. A field name appearing within the *print-list* of an AFTER CHANGE clause need not be explicitly included within the LIST statement. If the field name is not included, ENFORM effectively adds it with a NOPRINT clause.

Spacing Considerations

By default the AFTER CHANGE clause information begins printing in the same column position as the leftmost report column. Using SPACE or TAB clauses as the first element of the *print-list* overrides this default. SPACE or TAB clauses can also appear anywhere within the print list. In the following example, the SPACE clauses causes the two literals to be separated by five spaces:

```
AFTER CHANGE ON ordernum
  PRINT "*****" SPACE 5 "Orders for " ordernum,

*****     Orders for 122
```

If you specify either a SKIP clause or the slash symbol (/) within a *print-list*, ENFORM advances one or more lines before printing the rest of the AFTER CHANGE *print-list*. The number of lines advanced can be affected by one or more of the following: the digit (if any) following the keyword SKIP, the number of slashes specified, or the option variable @VSPACE. In the following example, the SKIP clause within the AFTER CHANGE clause causes two lines to be printed:

```
AFTER CHANGE ON regnum PRINT "Beginning of Report for"
  SKIP "Region " regnum,

Beginning of Report for
Region 1
```

Using the CENTER clause following the *print-list* of an AFTER CHANGE clause centers the information within the leftmost and rightmost columns of the report.

The CENTER, Option Variables, SKIP, SPACE, and TAB clauses are described in this section.

ASCD AND DESC CLAUSES

The ASCD and DESC clauses sort target-records, in ascending or descending order respectively, according to the value of the specified field. The syntax of the ASCD and DESC clauses is:

```
{ASCD}  target-item
{DESC}
```

where

target-item

is the name of a field from an input record which serves as a sort key for the target records.

The ASCD and DESC clauses do not group field values. If you want field values both grouped and sorted, use the BY or BY DESC clauses described in this section.

When more than one ASCD, BY, BY DESC, or DESC clause is specified in a LIST or FIND statement, ENFORM uses a major to minor sorting precedence. ENFORM determines the sorting precedence by the order in which the ASCD or DESC clauses appear in the LIST or FIND statement. The first sort clause has the highest priority, the next one second priority, down to the last specified clause.

The following ASCD clause indicates that the target records are to be sorted on the value of *partnum*:

```
ASCD partnum,
```

AS CLAUSE

The AS clause specifies a display format for printing a target-item or by-item. The syntax of the AS clause is:

```
{ 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 a target-item.

nonrepeatable-edit-descriptors

specify some general ways *report-items* are to be printed. *Nonrepeatable-edit-descriptors* should not be specified without a *repeatable-edit descriptor. Nonrepeatable-edit-descriptors* are:

nP	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* 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.

The AS clause identifies how an target-item or by-item is printed in the current report. When the AS clause is not specified, ENFORM uses the display format from one of these sources:

- the display format specified in the dictionary for the field
- the data picture specified in the dictionary for the field
- the display format in the AS clause of a DECLARE statement
- the internal format specified in a PARAM statement

The AS clause allows you to utilize some of the capabilities of the GUARDIAN Operating System Formatter. When you specify a display-format, ENFORM passes your format specifications to the Formatter. Refer to the *GUARDIAN Operating System Programming Manual* for more information about the Formatter.

Repeatable Edit Descriptors

Repeatable edit descriptors specify the conversion of data values by the Formatter for the target-item or by-item being printed. The word repeatable has no real meaning in the context of ENFORM; the word is used to simplify cross references to the FORMATTER terminology. Currently, ENFORM makes only single target-items or by-items available for modification by edit descriptors.

ALPHANUMERIC EDIT DESCRIPTOR. The alphanumeric edit descriptor specifies the target-item or by-item is to be printed using alphanumeric display format. The syntax is:

```
A [w]
```

where

w

is an unsigned integer that specifies the width in characters of the value of the target-item or by-item to be printed. The maximum value for *w* is 255 characters.

If you specify *w*, ENFORM prints the number of characters specified for the value of the target-item or by-item. If you omit *w*, ENFORM prints the actual number of characters in the value of the target-item or by-item. In either case, ENFORM prints the value of the target-item or by-item left-justified with blank fill.

If *w* is too small for the value of the target-item or by-item, an overflow condition occurs and ENFORM truncates the value.

Examples of the Alphanumeric Edit Descriptors. The following examples show the affect of the A edit descriptor:

```
Format       Item Value   Printed Item
------       ----------   ------------
A            WORD         WORD
A4           WORD         WORD
A3           WORD         WOR
"[LJ] A3"    WORD         WOR
"[RJ] A3"    WORD         ORD
```

INTEGER EDIT DESCRIPTOR. The integer edit descriptor specifies an integer display format. The syntax is:

```
Iw [.m]
```

where

w

is an unsigned integer that specifies the width of the output.

m

is an unsigned integer that specifies the number of digits that must be printed. The value of *m* must not exceed the value of *w*.

When you modify an target-item or by-item with an integer edit descriptor, ENFORM prints the value right-justified with blank fill. ENFORM always prints at least one digit for the value of such a target-item or by-item. If you specify *m*, ENFORM prints the number of digits specified, using leading zeroes if necessary.

Note that if the value to be printed is zero and you specify Iw.0, ENFORM prints blanks.

Examples of the Integer Edit Descriptor. The following examples show the effect of the integer edit descriptor.

```
Format   Item Value   Printed Item
------   ----------   ------------
I7       100              100
I7.2      -1              -01
I7.6     100           000100
I7.6      -1          -000001
```

FIXED FORMAT EDIT DESCRIPTOR. The fixed format edit descriptor specifies a fixed point display format. The syntax is:

```
Fw.d [.m]
```

where

w

is an unsigned integer that defines the total width of the target-item or by-item value.

d

is an unsigned integer that defines the number of digits that are to appear to the right of the decimal point.

m

is an unsigned integer that defines the number of digits that are to appear to the left of the decimal point.

When you specify the fixed format edit descriptor, ENFORM prints the value of the target-item or by-item right-justified with leading blanks if necessary. If the value is negative, ENFORM prints a minus sign before the first digit. Both the minus sign and the decimal point occupy one position of the format; therefore, *w* must be wide enough to accomodate the total size of the output field including the minus sign and the decimal point. if *w* is not wide enough, an overflow condition occurs.

If you specify *m*, ENFORM prints that number of digits using leading zeros if necessary.

Examples of the Fixed Format Edit Descriptor. The following examples show the effect of the fixed format edit descriptor.

```
Format             Item Value    Printed Item
------             ----------    ------------
F10.4              123.4567        123.4567
F10.4              0.000123          0.0001
F10.4.3            -4.56789       -004.5679
"[FL'*'] F10.2"    123.4567       ****123.46
```

MASK EDIT DESCRIPTOR. The mask edit descriptor specifies a display format according to a template. The syntax is:

```
M mask
```

where

mask

is a set of symbols or characters enclosed within quotation marks (" "), apostrophes (' '), or angle brackets (< >). The symbols in a mask that serve a special function are:

`Z` is a digit selector. *Z* specifies that if no digit exists, zeros are suppressed. *Z* can be used with alphanumeric or numeric data. Note: only a capitol *Z* is the special symbol.

`9` is a digit selector. If no digit exists, zeros are printed; used for numeric data only.

`V` indicates decimal alignment for the display format. When *V* is specified, the decimal point is not printed. Note: only a capitol *V* is the special symbol.

`.` indicates decimal alignment for the display format. When the symbol . (period) is used, the decimal point is printed.

Numeric Values. If the mask specified for a numeric value is too small, an overflow condition occurs.

The special symbols *Z* and *9* describe numeric values. If the digit selector is a *9*, ENFORM prints the corresponding data digit. If the digit selector is a *Z*, ENFORM prints the corresponding data digit unless it is a leading or trailing zero. In this case, ENFORM blank fills the position held by the zero.

Decimal point location can be indicated for numeric values by using either the symbol . or the symbol *V*. When the symbol . is used, a decimal point prints. When the symbol *V* is used, it indicates only decimal alignment and no decimal point prints. If neither the symbol. nor the symbol *V* is specified, ENFORM assumes the decimal point is the rightmost character of the entire *mask*.

Alphabetic Values. Alphabetic values must be specified by the symbol *Z* in a *mask* edit descriptor.

Examples of the Mask Edit Descriptor. The following examples show the affect of the mask edit descriptor.

```
Format           Item Value   Printed Item
-----            ----------   ------------
M"99/99/99"       103179       10/31/79
M'Z,ZZ9.99'       32.009          32.01
M<Z,ZZZ>          666              666
M<9,999>          666            0,666
M<9,999>          66666          *****
M<$ZZZ,ZZ9.99>    92000.00       $ 92,000.00
```

A query with these AS clauses:

```
amount AS M<$ZZ,ZZ9>,
date AS  M<Z9/Z9/99>,
district AS A8,
telephone AS M<(999) 999-9999>
```

for these target-item values:

```
Amount      := 9758  21573  15532
Date        := 031777  091779  090579
District    :=  West  Midwest  South
Telephone   := 2135296800  2162296270  4047298400
```

produces this report:

```
$ 9,758   3/17/77   West       (213) 529-6800
$21,573   9/17/77   Midwest    (216) 229-6270
$15,532   9/ 5/79   South      (404) 729-8400
```

Nonrepeatable Edit Descriptors

Nonrepeatable edit descriptors indicate some ways target-items or by-items are to be printed. Values described by the nonrepeatable edit descriptors do not require data conversion by the Formatter; however, the Formatter does process these values. When nonrepeatable edit descriptors are specified with modifiers or decorations, both the nonrepeatable edit descriptor and the repeatable edit descriptor must be enclosed in parentheses. The types of nonrepeatable edit descriptors are:

- Scale factor edit descriptor
- Optional plus edit descriptor

SCALE FACTOR EDIT DESCRIPTOR. The scale factor edit descriptor specifies a scale of 10**n for a fixed point (F) number. The value printed equals the internally represented number multiplied by 10**n. The syntax is:

```
nP
```

where

n

is the exponent for the scale factor (10**n).

P

is the implied decimal point of the number.

Examples of the Scale Factor Edit Descriptor. The following examples show the effect of the scale factor edit descriptor.

```
Format        Item Value   Printed Item
------        ----------   ------------
"2P F10.2"    100.00       10000.00
"-2P F10.2"   100.00           1.00
```

OPTIONAL PLUS EDIT DESCRIPTOR. The optional plus edit descriptors are used to control the printing of a plus (+) sign. The syntax is:

```
{S  }
{SS }
{SP }
```

where

S or SS

indicates no plus (+) sign is to be printed.

SP

indicates a plus (+) sign is to be printed.

When an object-item or by-item with a positive value is printed, ENFORM does not normally precede the value with a plus (+) sign. Specify *SP* when you want ENFORM to precede a value with an optional plus sign.

Examples of the Optional Plus Edit Descriptor. The following examples show the effect of the optional plus edit descriptor.

```
Format        Item Value   Printed Item
------        ----------   ------------
"SP F10.2"     123.00         +123.00
F10.2          123.00          123.00
"SP F10.2"    -123.00         -123.00
"SP F10.2"     000.00             .00
```

Modifiers

Modifiers alter the normal effect of edit descriptors. A modifier must immediately precede the edit descriptor it modifies. Modifiers are enclosed within brackets ([ ]) and separated by commas.

Table 5-2 indicates which modifiers can be used with which edit descriptors. An X indicates the combination is permitted.

Table 5-2. Permissible Modifiers and Edit Descriptors

Modifiers	Edit Descriptors			
	A	F	I	M
BZ, BN	X	X	X	X
LJ, RJ	X			
OC		X	X	X
FL	X	X	X	X
SS		X		X

FIELD BLANKING MODIFIERS. Field blanking modifiers indicate under what circumstances to print blanks. The syntax is:

```
{BN}
{BZ}
```

where

BN

prints a blank field if value is null.

BZ

prints a blank field if value is zero.

Although most edit descriptors cause a minimum number of characters to be printed regardless of the value of the field, a field blanking modifier causes the entire field to be filled with blanks if the specified condition is met.

Examples of the Field Blanking Modifiers. The following examples show the effect of the field blanking modifiers.

```
Format          Item Value    Printed Item
------          ----------    ------------
"[BZ] F10.2"    .00
"[BN] F10.2"    null value
"[BN] F10.2"    100.00             100.00
```

FILL CHARACTER MODIFIER. The fill character modifier specifies the fill character that is used in the current display format. The default fill character is a blank. The syntax is:

```
FL char
```

where

char

is a single ASCII character enclosed within apostrophes (' ').

A fill character prints in each appropriate character position when one of the following occurs: alphanumeric data contains fewer characters than the field specified by the alphanumeric edit descriptor, leading zero suppression is performed, or embedded text in an mask edit descriptor is not printed because its neighboring digits are not printed.

Examples of the Fill Character Modifier. The following examples show the affect of the fill character modifier.

```
Format                    Item Value   Printed Item
------                    ----------   ------------
"[FL'.'] A10"             THEN         THEN......
"[RJ,FL'>'] A10"          HERE         >>>>>>HERE
"[FL'*'] M<$ZZ,ZZ9.99>"   127.39       $***127.39
```

OVERFLOW CHARACTER MODIFIER. The overflow character modifier temporarily overrides the global default overflow indicator for the current display format. The overflow indicator is printed when a value exceeds the width specified in the display format. The syntax is:

```
OC char

where

  char

    is a single ASCII character enclosed within apostrophes (' ').
```

The overflow condition occurs if there are more characters to be printed than the display format specifies. When the overflow condition occurs, ENFORM prints the overflow character. The default overflow character is an asterisk (*). The overflow character modifier allows you to temporarily substitute another ASCII character for the asterisk (*). The overflow character modifier applies only to the display format where it is specified. The overflow modifier does not apply to the alphanumeric (A) edit descriptor.

If you want to change the default overflow character for all display formats in the current session, use the @OVERFLOW option variable described in this section.

Examples of the Overflow Character Modifier. The following examples show the effect of the overflow character modifier.

```
Format           Item Value   Printed Item
------           ----------   ------------
"[OC'!'] I2"     100          !!
"[OC'!'] F5.2"   100000.00    !!!!!
```

JUSTIFICATION MODIFIERS. The justification modifiers specify right or left justification for the values of an alphanumeric target-item or by-item. The syntax is:

```
{LJ}
{RJ}
```

where

LJ

specifies that an item is to be printed justified to the left of the specified display format width.

RJ

specifies that an item is to be printed justified to the right of the specified display format width.

The justification modifiers apply only to alphanumeric (A) edit descriptors. Alphanumeric target-items or by-items are normally left-justified. Padding of the item width automatically takes place. If the item value is wider than the width specified, truncation occurs.

Examples of the Justification Modifiers. The following examples show the effect of the justification modifiers.

```
Format       Item Value   Printed Item
------       ----------   ------------
"[RJ] A12"   HELLO              HELLO
A12          HELLO        HELLO
"[RJ] A2"    HELLO        LO
```

SYMBOL SUBSTITUTION MODIFIER. The symbol substitution modifier allows you to change the standard symbols ('9', 'V', 'Z', '.' and ',') used in edit descriptors. The syntax is:

```
SS pair-of-symbols
```

where

pair-of-symbols

are the standard and substitute symbols enclosed within apostrophes (' '). The standard symbol must appear first, followed by the substitute symbol.

The symbol substitution modifier allows you to replace the standard symbols used in edit descriptors. You can change the symbols for 9, Z, V, . and , in the mask (M) edit descriptor or change the symbol for decimal point (.) in the fixed point (F) edit descriptor.

The symbol substitution modifier is useful when you need to specify a character string, that is also a standard mask symbol. An example would be 9 in a mask for a date. Refer to the last example below.

Examples of the Symbol Substitution Modifier. The following examples show the effect of the symbol substitution modifier.

```
Format                      Item Value   Printed Item
------                      ----------   ------------
"[SS'.:'] F6.2"             12.45         12:45
"[SS'.:'] M<ZZZ.99>"        12.45         12:45
"[SS'.,'] F10.2"            12345.67       12345,67
"[SS'9X'] M<XX/XX/19XX>"    103179        10/31/1979
```

Decorations

Decorations specify character strings that can be printed along with the value of a target-item or by-item. Decorations also specify the conditions under which the character string is added, the location at which the character string is added, and whether the character string is added before normal formatting is done or after it is completed. The syntax of a decoration is:

```
condition location char-string
```

where

condition

is 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 one of the following:

- An absolute position n.
- F floating.
- P prior.

char-string

is one or more alphanumeric characters enclosed within apostrophes (' ').

The following rules apply to decorations:

- Separate multiple decorations by commas.
- Enclose decorations in brackets along with any modifiers.
- Specify decorations immediately before the edit descriptor they modify.
- Enclose decorations and the edit descriptors they modify within quotation marks (" ").

ENFORM evaluates decorations from left to right.

You are responsible for ensuring that the display format width is large enough to contain both the target-item or by-item value and the inserted character string.

CONDITIONS. The condition specifiers (negative, positive, zero, null, or overflow) indicate that a character string prints only when the specified condition occurs. A null condition takes precedence over negative, positive, and zero conditions. The overflow condition test is done after the other conditions are tested. Conditions specified for alphanumeric target-items or by-items can be positive or null only.

More than one condition specifier can exist for a decoration. Conditions are coded without separators.

LOCATION. The location specifier indicates where the character string prints in relation to the value of the target-item or by-item. The uses of the location specifiers are as follows:

- The *A* location specifier indicates the character string is to be printed starting in the absolute position *n*.
- The *F* specifier indicates the character string is to occupy the position or positions immediately to the left (for right-justified values) of the leftmost data character. If the value is left-justified, the *F* specifier indicates the character string occupies the position or positions immediately to the right of the rightmost data character.
- The *P* location specifier indicates that prior to normal formatting, the string is to be inserted in either the rightmost position (for right-justified values) or the leftmost position for left-justified values. The data values are shifted an appropriate number of positions.

PROCESSING ORDER. ENFORM processes decorations in the following order:

1. The data is tested to determine if it has a null value.
2. The data is tested to determine if it has a positive, negative, or zero value.
3. If the *P* location decoration is specified, the character string is added to the item value.
4. Normal formatting according to the edit descriptor (alphanumeric (A), integer (I), or fixed point (F) is performed.
5. Decorations for alphanumeric and fixed point edit descriptors are performed.
6. The overflow condition is tested.

DEFAULT DECORATIONS. When decorations are not specified, ENFORM prints a negative value with a preceding negative (–) sign. In other words, [MF'–'] is assumed.

If a decoration is specified that tests for a positive value, such as [PF'+'], the default [MF'–'] no longer automatically applies. In this case, the negative condition must be explicitly indicated if you want ENFORM to print the negative sign.

When an overflow condition occurs, ENFORM replaces the value by enough asterisks (*) to fill the field. In other words, [OA1'*****...***'] is assumed. The OC char modifier temporarily overrides the default asterisk (*) overflow character.

Examples of Decorations. Possible decorations and their meanings are:

MAn char-string	if value is negative, print char-string in position n.
MF char-string	if value is negative, print char-string immediately to the left of right-justified value, immediately to the right of left-justified value.
MP char-string	if value is negative, print char-string immediately to the right of value.
NAn char-string	if value is null, print char-string in position n.
NF char-string	if value is null, print char-string immediately to the left of right-justified value; immediately to the right of left-justified value.
NP char-string	if value is null, print char-string immediately to the right of value.
PAn char-string	if value is positive, print char-string in position n.
PF char-string	if value is positive, print char-string immediately to the left of value.
PP char-string	if value is positive, print char-string immediately to the right of value.
ZAn char-string	if value is zero, print char-string in position n.
ZF char-string	if value is zero, print char-string immediately to the left of right-justified value, immediately to the right of left-justified value.
ZP char-string	if value is zero, print char-string immediately to the right of value.
OAn char-string	if overflow condition occurs, print char-string in position n.

The following are examples of decorations:

```
Format                              Item Value     Printed Item
------                              ----------     ------------
"[MF'<',MP'>',ZPP'b'] F12.2"         1000.00          1,000.00
"[MF'<',MP'>',ZPP'b'] F12.2"        -1000.00         <1,000.00>

"[MA1'CR',MPF'$'] F12.2"             1000.00          $1,000.00
"[MA1'CR',MPF'$'] F12.2"             -100.00       CR   $100.00
"[MA1'CR',MPF'$'] F12.2"                0.00             0.00

"[OA1'**overflow**'] F12.2"          1000000.00     1000000000
"[OA1'**overflow**'] F12.2"       1000000000.00     **overflow**

"[ZPA2'+'] I8"                       -10               10
"[ZPA2'+'] I8"                       100             +   100
"[ZPA2'+'] I8"                       0               +     0
```

AS DATE CLAUSE

The AS DATE clause allows you to specify the display format for printing a date. The syntax of the AS DATE clause is:

```
date-in-internal-format AS DATE { *              }
                                { display-format }
```

where

`date-in-internal-format`

is the name of a variable or field that contains a date in internal format. The option variable @DATE can be specified to give the current date in internal format.

*

specifies the default display format. The default display format specifies the date in the form month/day/year. It is comparable to the *display-format* "M2/D2/Y2"

`display-format`

is the format for printing a date. *Display-format* must be specified within quotation marks (" "). *Display-format* can include date keywords, and other characters such as blanks, commas, hyphens, or slashes. The date keywords are:

M	specifies a month.
D	specifies a day.
Y	specifies a year.
A	abbrieviates or completely spells out the month or day. If n is specified with the keyword A, only n letters are displayed.
B	suppresses leading zeros.
O	abbrieviates or completely spells out the number corresponding to the day. If n is specified with the keyword O, only n letters are displayed.
n	an integer that specifies the number of characters (1-3) or numbers (2-4) to be printed.

When you want a date printed, the date must be in internal format. For instructions on how to convert a date to internal format, refer to the JULIAN-DATE or TIMESTAMP-DATE clause in this section. The date can be a target-item within a LIST statement or any element that can be modified by an AS DATE clause within a print list.

Default Display Format

ENFORM's default date *display-format*, "M2/D2/Y2", might handle most of the date formatting required. Change the default format by redefining the @DATE-FORMAT option variable discussed in this section.

Examples of Date Display Formats

These date keywords produce the following:

```
MA3    JANUARY,FEBRUARY,...,DECEMBER
MA3    JAN,FEB,...,DEC
M2     01,02,...,12
M      1,2,...,12
MB2    1, 2,...,12
DA     MONDAY,TUESDAY,...,SUNDAY
DA3    MON,TUE,...,SUN
D2     01,02,...,31
DB2    1, 2,...,31
D3     001,002,...,366
DB3    1,  2,...,366
DO2    1ST, 2ND, ... 31ST
DAO    FIRST, SECOND, ... ,THIRTY-FIRST
Y2     00,01, ...,76,77,78...
YB2    0, 1, ...,76,77,78...
Y4     1900,1901,...,1976,1977,1978...
```

AS TIME CLAUSE

The AS TIME clause allows you to specify the display format for printing a time. The syntax of the AS TIME clause is:

```
time-in-internal-format AS TIME {*              }
                                {display-format}
```

where

`time-in-internal-format`

is the name of a variable or field which contains a time in internal format.

`*`

specifies the default display format which specifies the hour, minute, and second as HB2:MB2:SB2 (see the following description of *display-format*). When the default display format is used military time is displayed.

`display-format`

is the format for printing the time. *Display-format* must be specified within quotation marks (" "). *Display-format* can be specified by using time keywords and other characters such as blanks, commas, hyphens, or slashes. In addition alphanumeric characters enclosed in apostrophes (' ') can be embedded within *display-format*. The time keywords are:

`H`	specifies an hour.
`M`	specifies a minute.
`S`	specifies a second.
`P`	expresses the hour as modulo 12 with AM or PM.
`B`	suppresses leading zeros.
`n`	an integer digit specifying the number of digits (1 or 2) printed.

If you want ENFORM to print the time, it must be in internal format. For instructions on how to convert a time to internal format, refer to the TIMESTAMP-TIME clause in this section. The time can be a target-item within a LIST statement or any element that can be modified by an AS TIME clause within a print list.

Default Time Display Format

ENFORM's default time format, "HB2:MB2:SB2", might handle most of the time formatting required. Change the default format by redefining the @TIME-FORMAT option variable discussed in this section.

Examples of the Time Display Format

These time keywords produce the following:

```
HB2      1, 2,....,24
HP2     01,02,....,12  AM or PM
HPB2     1, 2,....,12  AM or PM
M2      00,01,....,59
MB2      0, 1,....,59
S2      00,01,....,59
SB2      0, 1,....,59
```

AT END PRINT CLAUSE

The AT END PRINT clause prints information at the end of the current report. This clause is an optional part of the LIST statement. The syntax of the AT END PRINT clause is:

```
AT END PRINT print-list [ CENTER ]
```

where

print-list

contains any combination of literals, FORM, SKIP, SPACE and TAB clauses. *Print-list* can also contain the following elements that can be modified by AS, AS DATE or AS TIME clauses: field names, arithmetic expressions, IF/THEN/ELSE expressions, System Variable clauses, JULIAN-DATE clauses, TIMESTAMP-DATE clauses, TIMESTAMP-TIME clauses, user variables, or parameter names.

Clauses that can be used in a *print-list* are described in this section. The other elements are described in Section 3.

Specifying a Field Name Within an AT END PRINT Clause

If you specify a field name within the *print-list* of an AT END PRINT clause, ENFORM prints the same value as in the last row of the report. A field name appearing within the *print-list* of an AT END PRINT clause need not be explicitly included within the associated LIST statement. If the field name is not included, ENFORM in effect adds the field to the LIST statement with a NOPRINT clause.

Spacing Considerations

By default, the information you specify in the *print-list* of an AT END PRINT clause begins printing in the same column position as the leftmost column of the report. Using the SPACE or TAB clause as the first element of the *print-list* overrides the default. SPACE or TAB clauses can be used anywhere within the *print-list*. In the following example, the SPACE clause causes the two literals to be separated by 15 spaces:

```
AT END PRINT "Report" SPACE 15 "Total Sales",

Report               Total Sales
```

If you specify either a SKIP clause or the slash symbol (/) within a *print-list*, ENFORM advances one or more lines before printing the rest of the AT END *print-list*. The number of lines advanced can be affected by one or more of the following: the digit (if any) following the keyword SKIP, the number of slashes specified, or the option variable @VSPACE. In the following example, the SKIP clause in the *print-list* causes ENFORM to print two lines:

```
AT END PRINT "End of Report for" SKIP "Region " regnum,

End of Report for
Region 1
```

Using the FORM clause within the *print-list* causes ENFORM to start a new page, increment the page number, and continue with the rest of the *print-list*.

Using the CENTER clause following the *print-list* centers the information within the leftmost and rightmost columns of the report.

The CENTER, Option Variable, SKIP, SPACE, and TAB clauses are described in this section.

AT END Information for Current Report or All Reports

The optional AT END PRINT clause prints information only for the current report. This clause temporarily overrides the session-wide AT END statement. If you want to print information at the end of all subsequent reports in the current session, use the AT END statement.

Overriding Session-Wide AT END Information

Temporarily override session-wide AT END information by specifying the AT END PRINT clause with " " for the *print-list* parameter.

AT START PRINT CLAUSE

The AT START PRINT clause allows you to specify information to be printed just before the first set of column headings for the current report. This clause is an optional part of the LIST statement. The syntax of the AT START PRINT clause is:

```
AT START PRINT print-list [ CENTER ]
```

where

print-list

contains any combination of literals, FORM, SKIP, SPACE or TAB clauses. A *print-list* can also contain the following elements that can be modified by AS, AS DATE, or AS TIME clauses: field names, arithmetic expressions, IF/THEN/ELSE expressions, System Variable clauses, JULIAN-DATE clauses, TIMESTAMP-DATE clauses, TIMESTAMP-TIME clauses, user variables, or parameter names.

The clauses that can appear in a *print-list* are described in this section. The other elements are described in Section 3.

The AT START PRINT clause differs from both the TITLE and SUBTITLE statements and clauses in that a title or subtitle is printed on every page of a report while the AT START PRINT information is printed only on the first page of a report. The information supplied in an AT START PRINT clause prints after either a title or a subtitle.

Specifying a Field Name in an AT START PRINT Clause

If you specify a field name within a *print-list* of an AT START PRINT clause, ENFORM prints the same field value as in the first row of the report. A field name appearing within the print list of a AT START PRINT clause need not be explicitly included within the associated LIST statement. If the field name is not included, ENFORM in effect adds it with a NOPRINT clause.

Spacing Considerations

By default the AT START PRINT information begins printing in the same column position as the leftmost report column. Using SPACE or TAB clauses as the first element of the *print-list* overrides this default. SPACE or TAB clauses can also appear anywhere within the *print-list*. In the following example the SPACE clause causes the two literals to be separated by 15 spaces:

```
AT START PRINT "Report" SPACE 15 "Total Sales",

Report                Total Sales
```

If you specify either a SKIP clause or the slash symbol (/) within a *print-list*, ENFORM advances one or more lines before printing the rest of the *print-list*. The number of lines advanced can be affected by one or more of the following: the digit (if any) following the keyword SKIP, the number of slashes specified, or the option variable @VSPACE. In the following example, the SKIP clause causes two lines to be printed:

```
AT START PRINT "End of Report for" SKIP "Region " regnum,

End of Report for
Region 1
```

The use of the FORM clause within a *print-list* causes ENFORM to start a new page, increment the page number, and continue with the rest of the *print-list*.

Using the CENTER clause following the *print-list* of the AT START PRINT clause centers the information within the leftmost and rightmost columns of the report.

The CENTER, Option Variables, SKIP, SPACE, and TAB clauses are described in this section.

AT START Information for Current Report or All Reports

An AT START PRINT clause only prints information for the report generated by the associated LIST statement. It temporarily overrides the session-wide AT START statement. If you want to print the same information just before the first set of column headings for all subsequent reports in the current session, use the AT START statement.

Overriding Session-Wide AT START Information

Temporarily override the session-wide AT START statement by using the AT START PRINT clause with " " for the *print-list* parameter.

BEFORE CHANGE CLAUSE

The BEFORE CHANGE clause allows you to specify information to be printed following the records for each group for the current report. The BEFORE CHANGE clause is an optional part of the LIST statement. The syntax of the BEFORE CHANGE clause is:

```
BEFORE CHANGE [ ON ] by-item PRINT print-list [ CENTER ]
```

where

by-item

is the name of a field grouped by a BY or BY DESC clause.

print-list

is any combination of literals, FORM, SKIP, SPACE or TAB clauses. A *print-list* can also contain the following elements that can be modified by AS, AS DATE, or AS TIME clauses: field names, arithmetic expressions, IF/THEN/ELSE expressions, System Variable clauses, JULIAN-DATE clauses, TIMESTAMP-DATE clauses, TIMESTAMP-TIME clauses, user variables, or parameter names.

The clauses that can be used in a *print-list* are described in this section. The other elements are described in Section 3.

The keyword BEFORE in the BEFORE CHANGE clause refers to the values printed, not to the location of the printed information. For the exact location of where the BEFORE CHANGE information is printed within a report, refer to the *ENFORM Users Guide*.

When more than one BEFORE CHANGE clause is specified, ENFORM prints the BEFORE CHANGE information in the order in which the BEFORE CHANGE clauses are entered.

Specifying a Field Name Within a BEFORE CHANGE Clause

If you specify a field name within a *print-list* of a BEFORE CHANGE clause, ENFORM prints the same field value in the last row of the previous group; that is, before the value changes. A field name appearing within the *print-list* of a BEFORE CHANGE clause need not be explicitly included within the LIST statement. If the field name is not included, ENFORM in effect adds it with a NOPRINT clause.

Spacing Considerations

By default the BEFORE CHANGE clause information begins printing in the same column position as the leftmost report column. Using SPACE or TAB clauses as the first element of the print list overrides this default. SPACE or TAB clauses can also appear anywhere within the *print-list*. In the following example, the SPACE clause causes the two literals to be separated by five spaces:

```
BEFORE CHANGE ON ordernum
  PRINT "*****" SPACE 5 "Orders for " ordernum,

*****     Orders for 122
```

If you specify either a SKIP clause or the symbol / (slash) within a *print-list*, ENFORM advances one or more lines before printing the rest of the BEFORE CHANGE *print-list*. The number of lines advanced can be affected by one or more of the following: the digit (if any) following the keyword SKIP, the number of slashes specified, or the option variable @VSPACE. In the following example, the SKIP clause causes two lines to be printed:

```
BEFORE CHANGE ON regnum PRINT "Begin of Report for"
  SKIP "Region " regnum,

Begin of Report for
Region 1
```

Using the CENTER clause following the *print-list* of a BEFORE CHANGE clause centers the information within the leftmost and rightmost columns of the report.

The CENTER, Option Variable, SKIP, SPACE, and TAB clauses are described in in this section.

BY AND BY DESC CLAUSES

The BY and BY DESC clauses group and sort target-records according to the value of a specified field. The syntax of the BY and BY DESC clauses is:

```
{ BY      } by-item
{ BY DESC }
```

where

by-item

is the name of a field from an input record whose values are to be used to group and sort target-records.

BY and BY DESC clauses group and sort records according to the field values. The BY clause sorts the records in ascending order according to the value of the specified field; the BY DESC clause sorts in descending order. The following example groups and sorts the partnum field:

```
BY partnum,
```

When more than one sort is specified, ENFORM uses a major to minor sort precedence. ENFORM determines the sort precedence by the order in which a BY or BY DESC clause appears in a LIST or FIND statement. The first BY or BY DESC clause has the highest priority, the next one second priority, down to the last BY or BY DESC clause.

When a BY or BY DESC clause is used with a LIST statement, ENFORM prints only the first instance of a grouped value in a report. When a BY or BY DESC clause is used with a FIND statement, ENFORM writes all of the values of a grouped value to the physical output file or transmits all of the values to the host language program.

Several clauses require that a field be grouped by a preceding BY or BY DESC clause. Refer to the CUM, PCT, SUBTOTAL, AFTER CHANGE and BEFORE CHANGE clauses in this section and Target Aggregates in Section 3.

CENTER CLAUSE

The CENTER clause centers an object within its context. The syntax of the CENTER clause is:

```
( {target-item} CENTER )
( {by-item   }        )
(                     )
(  CENTER ALL         )
```

where

target-item

is a record name, a field name, a string literal, a predefined aggregate, a user aggregate, an arithmetic expression, an IF/THEN/ELSE expression, a user variable, or a System Variable clause.

by-item

is the name of a field grouped by a BY or BY DESC clause.

Centering Single Report Items

The CENTER clause causes a *target-item* or *by-item* to be centered under its column heading. Numeric data is normally right-justified; alphabetic data is normally left-justified. *Target-items* and *by-items* are centered based on the width for which they are formatted, not their actual values. If the space divides unevenly, ENFORM places the extra space to the right of the item.

Centering All Report Items

The CENTER ALL clause causes all elements in a LIST statement to be centered under their headings. It must follow the WHERE clause in the LIST statement. Refer to the syntax of the LIST statement in Section 4 for the relative locations of the clauses within a LIST statement.

Centering a Print List

The CENTER clause following the print list of an AT END statement or clause, an AT START statement or clause, an AFTER CHANGE clause, a BEFORE CHANGE clause, a FOOTING statement or clause, a SUBTITLE statement or clause, a SUBFOOTING statement or clause, or a TITLE statement or clause, centers the print list information within the leftmost and rightmost columns of the report.

CUM CLAUSE

The CUM clause allows you to specify printing of a running total for a numeric target-item either for all the instances of the target-item or for the instances of the target-item grouped within the each value of a by-item. The syntax of the CUM clause is:

```
target-item  CUM [OVER ALL    ]
                 [OVER by-item]
```

where

target-item

is a record name, a field name, a numeric literal, a predefined aggregate, a user aggregate, an arithmetic expression, an IF/THEN/ELSE expression, a user variable, or a System Variable clause. The data type of a target-item used in a CUM clause must be numeric.

by-item

is the name of a field grouped by a BY or BY DESC clause.

CUM With OVER ALL

When you specify the CUM OVER ALL clause, ENFORM prints a running total in place of each value of the numeric *target-item*. When you specify only CUM, ENFORM assumes CUM OVER ALL.

CUM With OVER

When you specify the CUM OVER *by-item* clause, ENFORM prints a running total for the instances of the numeric *target-item* within the *by-item* in place of the value of the numeric *target-item*. The *by-item* must be previously defined by a BY or BY DESC clause. The running total begins anew each time the *by-item* value changes. The following example prints the running total of all parts for each location:

```
LIST BY location,
     partnum,
     inventory,
     inventory CUM OVER location;
```

LOCATION	Part Number	INVENTORY	CUM INVENTORY
L98	5103	8	8
	5502	6	14
V66	6603	40	40
...	...	...	...

CUM Clause Used With User Variable

When a numeric *target-item* with a CUM clause is assigned to a user variable, ENFORM assigns the value to the user variable first, before the running total is calculated. When the user variable is referenced as a *target-item* element in a LIST statement or as an element within a LIST *target-item*, ENFORM uses the value of the user variable. When the user variable is referenced in a print list, ENFORM uses the value of the running total.

Restrictions

Note that you cannot combine the CUM clause with the PCT clause, the TOTAL clause, or the SUB-TOTAL clause.

FOOTING CLAUSE

The FOOTING clause allows you to specify information for printing at the bottom of each page for the current report. This clause is an optional part of the LIST statement. The syntax of the FOOTING clause is:

```
FOOTING print-list [ CENTER ]
```

where

print-list

can be any combination of literals, FORM, SKIP, SPACE, or TAB clauses. A *print-list* can contain the following elements that can be modified by AS, AS DATE, or AS TIME clauses: field names, arithmetic expressions, IF/THEN/ELSE expressions, System Variable clauses, JULIAN-DATE clauses, TIMESTAMP-DATE clauses, TIMESTAMP-TIME clauses, user variable names, or parameter names.

The clauses that can be used in a *print-list* are described in this section. The other elements are described in Section 3.

Specifying a Field Name Within a FOOTING Clause

If you specify a field name within a *print-list* of a FOOTING clause, ENFORM prints the same field value as in the last row of data on the current report page. A field name appearing within the FOOTING clause need not be explicitly included within the LIST statement. If the field name is not included, ENFORM effectively adds it to the LIST statement with a NOPRINT clause.

Spacing Considerations

By default the footing begins printing in the same column position as the leftmost report column. Using SPACE or TAB clauses as the first element of the *print-list* overrides this default. SPACE or TAB clauses can also appear anywhere within the *print-list*. In the following example, the SPACE clause causes the two literals to be separated by 15 spaces:

```
FOOTING "Inventory" SPACE 15 "Parts in Stock",
```

The following footing is printed on the next report:

```
Inventory               Parts in Stock
```

If you specify either a SKIP clause or the slash symbol (/) within a *print-list*, ENFORM advances one or more lines before printing the rest of the FOOTING *print-list*. The number of lines advanced can be affected by one or more of the following: the digit (if any) following the keyword SKIP, the number of slashes specified, or the option variable @VSPACE. In the following example, the SKIP clause causes two lines to be printed:

```
FOOTING "Report 2-A" SKIP "Total Sales",
```

The following footing prints on the next report:

```
Report 2-A
Total Sales
```

Using a FORM clause within a FOOTING statement forces a new page. Printing continues with the remainder of the FOOTING *print-list*, starting at the top of the next physical page. The page number remains the same. A single logical page can span multiple physical pages, such that a TITLE can appear on one page, the data on the next, and a FOOTING on the next. The same page number applies to all physical pages in the logical page.

Using the CENTER clause centers the footing within the leftmost and rightmost columns of the report.

The CENTER, Option Variables, SKIP, SPACE, and TAB clauses are described in this section.

Footing for Current Report or All Reports

A FOOTING clause within a LIST statement prints a footing only for the current report. A FOOTING clause temporarily overrides a sessionwide FOOTING statement. A FOOTING statement prints a footing at the bottom of each page for all subsequent reports in the current session.

FORM CLAUSE

The FORM clause allows you to control when to skip to a new page. The syntax of the FORM clause is:

```
FORM [ number ]

where

  number

    is an unsigned integer.
```

FORM Clause With a By-item

When the FORM clause follows a field name that has been grouped with a BY or BY DESC clause, ENFORM starts a new page whenever the field value changes. If *number* is specified, ENFORM starts a new page only if there are fewer than that number of lines remaining on the current page. The following example shows the use of the FORM clause with a by-item:

```
BY regnum FORM,
```

FORM Clause With a Target-item

When the FORM clause precedes a target-item, ENFORM starts a new page each time the target-item is printed. When the FORM clause follows a target-item, ENFORM prints the target-item before starting a new page. The *number* parameter is not allowed when the FORM clause modifies a target-item.

FORM Clause Within a Print-list

The FORM clause can be part of a print list for an AFTER CHANGE, AT START PRINT, AT END PRINT, BEFORE CHANGE, FOOTING, SUBFOOTING, SUBTITLE, or TITLE clause or statement. ENFORM starts printing on a new page everytime the FORM clause is processed. When used with a print list, the FORM clause does not use the *number* parameter.

HEADING CLAUSE

The HEADING clause allows you to override the default column title for a target-item or by-item in a report. The HEADING clause also allows you to define a column title for target-items, such as arithmetic expressions, that do not have a default column title. The syntax of the HEADING clause is:

```
{ by-item     } HEADING "heading-string"
{ target-item }
```

where

by-item

is the name of a field that has been grouped by a BY or BY DESC clause

target-item

is a record name, a field name, a numeric literal, a predefined aggregate, a user aggregate, an arithmetic expression, an IF/THEN/ELSE expression, a user variable, a System Variable clause, or a JULIAN-DATE clause.

heading-string

is a string literal. Remember string literals must be enclosed within quotation marks (" ").

Default Headings

When the HEADING clause is not specified explicitly, ENFORM obtains a heading from one of these sources:

- the HEADING clause specified either in the dictionary or in a DECLARE statement.
- the field name specified in the dictionary or name of the user variable, aggregate, or table specified in a DECLARE statement

If you want to print more than one *target-item* or *by-item* under the same column heading, use a NOHEAD clause to prevent ENFORM from printing the unwanted heading. The NOHEAD clause is discussed later in this section.

Multiple Line Headings

Multiple line headings are created by using a slash (/) within the heading string.

Printing / in a Column Heading

Sometimes, the / character needs to appear within a column heading. In this case, use the SET statement described in Section 4 to change the @NEWLINE option variable from a / to a different character. After the new line character is redefined, the new special character can be used within the heading string instead of the / character. In the following example, the new line character is changed to a number sign (#) causing Region and Number to print on two lines, and the symbol / inside the print list is printed between Branch and Number.

```
SET @NEWLINE TO "#";
LIST regnum HEADING "Region#Number",
     branchnum HEADING "Branch/Number"/;
```

```
Region
Number     Branch/Number
------     -------------

     1                1

     1                2
```

Notice that when the / symbol appears outside of the heading clause, it causes ENFORM to advance 1 line before printing the next target-item. Changing the @NEWLINE character does not affect this use of the / symbol.

Heading for Subscripted Elements

When a single subscripted element is modified by a HEADING clause, ENFORM prints the specified heading. For example:

```
LIST month [ 3 ], HEADING "MARCH";
```

causes ENFORM to print:

```
MARCH
-----
  ...
```

When an element including a subscript range is modified by a HEADING clause, ENFORM includes the subscript in the specified heading. For example:

```
LIST month [ 1:3 ], HEADING "FIRST QUARTER";
```

causes ENFORM to print:

```
FIRST QUARTER   FIRST QUARTER   FIRST QUARTER
    [1]             [2]             [3]
-------------   -------------   -------------
         ...             ...             ...
```

If a HEADING clause is not included, ENFORM includes the subscript with the default heading. For example:

```
LIST month [ 3 ];
```

causes ENFORM to print:

```
MONTH
[ 3 ]
-----
  ...
```

Remember the default heading for a field name is either the heading declared in the dictionary or if no heading is so declared, the field name. The default heading for a user variable is either the heading defined in the DECLARE statement, or if no heading is defined, the user variable name.

INTERNAL CLAUSE

The INTERNAL clause allows you to specify the storage format for a user defined element. The syntax of the INTERNAL clause is:

```
INTERNAL internal-format
```

where

internal-format

is the format for storing the user-defined element. *Internal-format* can be:

An	alphanumeric, where *n* is the length.
In	integer, where *n* is the length.
Fw.d	fixed, where *w* is the number of digits and *d* is the number of decimal places.

The optional INTERNAL clause can appear in a PARAM or DECLARE statement. When the INTERNAL clause is not used, ENFORM stores user-defined elements (variables, aggregates, tables, and parameters) as 64-bit signed integers. The following INTERNAL clause specifies the *internal-format* as fixed. The element has a total length of 8 digits with 2 digits following the decimal point:

```
INTERNAL F8.2
```

JULIAN-DATE CONVERSION CLAUSE

The JULIAN-DATE Conversion clause allows you to specify translation of a date target-item into internal format. The syntax for the JULIAN-DATE Conversion clause is:

```
JULIAN-DATE ( year , month , day )
```

where

year

is the year in 4 digits.

month

is the month in 2 digits, 1-12.

day

is the day in 2 digits, 1-31.

Dates are a common part of data base records. From an information standpoint, dates need to be printed on a report in a form recognizable as a date, such as 05/15/72, Apr 1, 1979 or 06-01-1970. From an analytical standpoint, dates need to be stored in a form for use in calculations or expressions, such as 80 11 27. Dates used in calculations or expressions must be in an internal format. If the date is not in this internal format, it can be converted by using the JULIAN-DATE Conversion clause. The internal format represents a date as the number of days which have elapsed from an arbitrary date in the past.

Conversion to Internal Format

To change a date to internal format, use the JULIAN-DATE Conversion clause. For example, assume the data description entry of the day and year portion of a data base date is as follows:

```
05   date.
     10  yy              PIC "99".
     10  mm              PIC "99".
     10  dd              PIC "99".
```

The month (1-12), day (1-31), and year (4 digit number) can be passed to the JULIAN-DATE Conversion clause as follows:

```
JULIAN-DATE ((1900 + yy), mm, dd)
```

Notice the numeric literal 1900 was added to make a 4 digit year.

Gregorian dates must be converted to an internal date format when used for purposes other than printing on a report. When the date only needs to be printed, convert it with the JULIAN-DATE clause and then use an AS DATE Conversion clause. For example:

```
JULIAN-DATE ((1900 + yy), mm, dd ) AS DATE *
```

Alternatively, convert the date to an integer that can be formatted with an AS clause. For example:

```
date AS M<99-99-99>
```

To convert a date stored as a yearly Julian date, where the day of the year is relative to January 1 of that year, define each part of the date:

```
05   yearly-julian-date.
     10  day-of-year      PIC "999".
     10  current-year     PIC "99".
```

Then add day-of-year to the internal date for December 31 of the previous year:

```
(JULIAN-DATE ((1900 + (current-year - 1)), 12, 31) + day-of-year)
```

Display Format

To print a date in internal format on a report, convert the date to a display format with an AS DATE clause. The AS DATE clause is discussed in this section.

NOHEAD CLAUSE

The NOHEAD clause allows you to specify suppression of the printing of the column heading of a target-item or by-item. The syntax of the NOHEAD clause is:

```
( { target-item } NOHEAD )
( { by-item    }         )
(                        )
(   NOHEAD ALL           )
```

where

target-item

is a record name, a field name, a numeric literal, a predefined aggregate, a user aggregate, an arithmetic expression, an IF/THEN/ELSE expression, a user table name, a user variable, a System Variable clause, or a JULIAN-DATE clause.

by-item

is the name of a field grouped by a BY or BY DESC clause.

No Headings for Single Report Items

The NOHEAD clause following a LIST *target-item* suppresses the column heading for the *target-item*.

Sometimes it is undesirable to print a column heading for an item such as when you want to print more than one target-item under the same column heading. In the following example, the address, city, and state are all printed in the same column:

```
LIST suppnum,
    address / TAB 10,
    city NOHEAD / TAB 10,
    state NOHEAD;

SUPPNUM          ADDRESS
-------   ----------------------

      1   19333 VALLCO PARKWAY
          CUPERTINO
          CALIFORNIA
      2   2000 BAKER STREET
          IRVINE
          CALIFORNIA
```

No Headings for All Report Items

The NOHEAD ALL clause suppresses column headings for all the *target-items* specified in a LIST statement. It must follow the WHERE clause in the LIST statement. Refer to the syntax of the LIST statement in Section 4 for the relative locations of the clauses within a LIST statement.

NOPRINT CLAUSE

The NOPRINT clause allows you to specify suppression of the printing of a target-item or by-item and its associated column heading. The syntax of the NOPRINT clause is:

```
{ {target-item} NOPRINT }
{ {by-item    }         }
{                       }
{ NOPRINT ALL           }
```

where

target-item

is a record name, a field name, a numeric literal, a predefined aggregate, a user aggregate, an arithmetic expression, an IF/THEN/ELSE expression, a user table name, a user variable, a System Variable clause or a JULIAN-DATE clause.

by-item

is the name of a field grouped by a BY or BY DESC clause.

Suppress Single Report Items

The NOPRINT clause following a LIST *target-item* or *by-item* suppresses the printing of the *target-item* or *by-item* and its heading. The *target-item* or *by-item* can still be used for sorting, grouping, or calculations elsewhere in the report.

You can use the NOPRINT clause to suppress printing of a *target-item* or *by-item* within the body of a report. In the following example, the records are grouped by region, but the region number is not printed as a *by-item* in the report. Instead the region number is printed at the end of the group using a BEFORE CHANGE clause:

```
LIST BY regnum NOPRINT,
     branchnum,
     manager,
  BEFORE CHANGE ON regnum PRINT "Region " regnum;

        BRANCHNUM   MANAGER
        ---------   -------

                1        75
                2       129

        Region 1
```

Suppress All Report Items

The NOPRINT ALL clause suppresses the printing of all column and column headings. It must follow the WHERE clause in the LIST statement. Refer to the syntax of the LIST statement in Section 4 for the relative locations of the clauses within a LIST statement.

Title and summary information, using AT END, AT START, FOOTING, SUBFOOTING, SUBTITLE, TITLE statements and clauses, and AFTER CHANGE and BEFORE CHANGE clauses still appear in the report. This is useful for reports where only summary information is desired.

OPTION VARIABLE CLAUSES

The Option Variable clauses allow you to redefine the default values for several operational variables. Refer to the SET statement in Section 4. The Option Variables and their legal values are:

```
{ @BLANK-WHEN-ZERO }
{ @BREAK-KEY       }
{ @CENTER-PAGE     }
{ @HEADING         } TO { ON  }
{ @STATS           }    { OFF }
{ @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 string-literal

{ @DATE-FORMAT }     TO display-format
{ @TIME-FORMAT }
```

where

number

is an integer.

"character"

is a single ASCII character enclosed within quotation marks.

string-literal

is a string literal; remember string literals must be enclosed in quotation marks (" ").

display-format

specifies the default format for printing dates or times when AS DATE * and AS TIME * clauses are used; must be enclosed in quotation marks.

@BLANK-WHEN-ZERO

When set to *ON*, ENFORM suppresses the printing of target-item or by-item values of zero in reports. When set to *OFF*, zeros are printed. Default is *OFF*.

@BREAK-KEY

When set to *ON*, pressing the BREAK key while a query is executing, either terminates output (if output is being produced) or terminates the current query and returns you to the ENFORM prompt. Refer to Section 2 for more information. When set to *OFF*, pressing the BREAK key temporarily suspends output and returns you to the Command Interpreter prompt but it does not terminate processing. Output resumes when you enter the Command Interpreter PAUSE command. If the file specified in the IN option of the ENFORM command is not a terminal, ENFORM ignores the BREAK key regardless of the setting of *@BREAK-KEY*. Default is *ON*.

@CENTER-PAGE

When set to *ON*, ENFORM centers the report body on the page according to the listing device's maximum page width. If the *@MARGIN* Option Variable clause is also set, that margin value is added to the left after centering.

This clause does not center the TITLE, SUBTITLE, AT START, AT END, FOOTING, or SUBFOOTING information within the report body. In effect the entire report body is moved enough spaces to the right to cause it to be centered on the page.

When not set or set to *OFF* the report begins in column one. Default is *OFF*.

@COPIES

specifies how many copies to print. Default is 1.

@COST-TOLERANCE

can be set to a number between 1 and 8. Setting @COST-TOLERANCE prevents ENFORM from executing a strategy that is more expensive than the user wants. When a larger *number* is specified, the query processor performs more work than when a smaller number is specified.

This does not mean that an ENFORM session which meets critera for level 2 will always take less time to run than another ENFORM session which requires level 7. The execution time also depends on the amount of data that must be processed.

When not set or set to 0 (zero), ENFORM proceeds with whatever strategy it chooses.

If the strategy specified is exceeded, an error message is received and the ENFORM program stops.

→

The desired cost tolerance level can be defined as follows:

0 allows the ENFORM session to proceed with no cost limit.

1 keyed access is used on all files.

2 one file might require one full-file read.

3 more than one full-file read is required.

4 one file might be sorted once for a link.

5 two files might be sorted once for a link.

6 one file might be sorted more than once for a link.

7 two files might be sorted more than once for a link.

8 two or more files can require full-file read at least once, with no convenient strategy for doing the link available.

`@DATE-FORMAT`

specifies a default format for printing dates when AS DATE * is used. For date format specifications, refer to the AS DATE clause in this section. The default is "M2/D2/Y2".

`@DECIMAL`

When set to a single ASCII character, ENFORM prints that character for a decimal point. ENFORM also accepts the specified character in place of a decimal point in a numeric literal. The default value is the period character (.).

`@DISPLAY-COUNT`

determines how many lines to display on output device at one time. When *number* lines are displayed, ENFORM pauses. To continue the display, press the carriage return. Execution of the ENFORM program, while at the pause, can be aborted by:

- Entering two slashes (//).
- Pressing the CTRL and Y keys simultaneously.
- Pressing the BREAK key.

Control returns to ENFORM. To reset the display count so that the entire ENFORM program is printed, enter 0 for *number*. Specifying @DISPLAY-COUNT when using an ENFORM server (described in the *ENFORM Users Guide*) is inadvisable. The default value is 0.

`@HEADING`

When set to *ON*, ENFORM prints column headings in the report. When set to *OFF*, ENFORM does not print headings. *OFF* is equivalent to NOHEAD ALL for every report. The default is *ON*.

`@LINES`

specifies how many lines are to be printed per page. The default value is 60. →

@MARGIN

specifies the left margin size in columns for reports. The default value is 0 (zero).

@NEWLINE

The new line character can appear within a heading string, inside the quotation marks. It is used to specify multiple line column headings. The *@NEWLINE* Option Variable clause can be reset to any single ASCII character except circumflex (^) or hyphen (-). The default NEWLINE character is / (slash).

@NONPRINT-REPLACE

ENFORM prints the replacement character for values which do not have a printable character. When this clause is set to a character, that character is printed for values which do not have a printable character. The default value is *OFF*.

@OVERFLOW

When a value does not fit within its formatted width, ENFORM replaces the value with overflow characters. When this clause is set to a single ASCII character, ENFORM prints that character when overflow occurs. The default overflow character is an asterisk (*).

@PAGES

specifies a maximum number of pages to print per report. The default value is 32,767 pages.

@PRIMARY-EXTENT-SIZE

specifies the primary extent size of the output file.

When this clause is not set, ENFORM calculates an estimate of the primary and secondary extent sizes needed for each output file. The estimate could be too large, creating a file system error 43 unable to obtain disc space for file extent; or the estimate could be too small, producing more data than the file can hold. Each time an output file is full, ENFORM must create a new output file, with a larger extent size, and copy the data from the full file into the new file. This results in very inefficient processing.

@READS

specifies the maximum number of records to be returned from the data base per ENFORM program. This is a useful tool when debugging. For an idea of how many reads are required by a given ENFORM program, see the *@STATS* Option Variable clause. To remove a limit, set the *@READS* Option Variable to 0 (zero). The default value is unlimited.

@SECONDARY-EXTENT-SIZE

specifies the extent size of the output files. When this clause is not set, ENFORM uses the primary-extent size. →

@SPACE

specifies how many blanks appear between report columns. The default value is two blanks.

@STATS

When set to *ON*, ENFORM prints the statistics regarding the records after the ENFORM program is completed. ENFORM prints the following statistics:

FILENAME	physical file name of a set of records.
LEVEL READ	order this physical file was read in relation to all physical files read. The first file read is indicated by the number 1.
RECORDS READ	total number of logical records read from this physical file.
POSITIONS	total number of positioning operations performed to read this physical file.
STRATEGY COST	number, 1 through 8, used to represent the cost of the strategy ENFORM developed to handle the ENFORM program. For a definition of the cost numbers, refer to the *@COST-TOLERANCE* option variable.

The default value is *OFF*.

@SUBTOTAL-LABEL

specifies a character string which is to appear on the same line as and to the left of the subtotal. ENFORM prints the character string in the column of the group item over which the subtotal is computed. If the item is defined with the NOPRINT clause, ENFORM does not print the character string. If the character string does not fit in the column, ENFORM truncates it on the right. The character string can be from 1 to 15 ASCII characters, enclosed within quotation marks. The default value is an asterisk (*).

@SUMMARY-ONLY

causes a summary report to be produced. See the explanation of summary reports found with the LIST and FIND statements in Section 4.

The default value is *OFF*.

@TARGET-RECORDS

specifies the maximum number of records to be selected per ENFORM program. The default value is zero, meaning no limit. Setting @TARGET-RECORDS to *number* when your query contains BY, BY DESC, ASCD, or DESC clauses returns the first target-records produced by the query processor which are not necessarily the first records that appear in a full report. →

@TIME-FORMAT

specifies a default format enclosed within quotation marks for printing times when AS TIME * is used. For time format specifications, refer to the AS TIME clause in this section. The default value is "H2:M2:S2".

@UNDERLINE

specifies a single ASCII character used to underline headings and totals. To specify no underlining, set the *@UNDERLINE* Option Variable clause to blank, " ". The default character is "_".

@VSPACE

specifies how many lines are skipped before a report line when the SKIP clause is used. The default number is one.

@WARN

specifies when warning messages are to appear on the terminal. When set to *OFF*, warning messages do not appear. The default is *ON*.

@WIDTH

specifies the maximum width of the output. The maximum value is 132 characters.

Verify the device's width before using this clause. This clause is convenient for setting a report page width smaller than the device's default width. The default width is the width of the output device being used. The default value for unstructured disc files is 132.

PCT CLAUSE

The PCT clause prints the percentage of the grand total for a numeric target-item, based either on a total figure for all instances of the target-item or a total for the instances of the target-item grouped over a by-item. The syntax of the PCT clause is:

```
target-item     PCT { OVER ALL    }
                    { OVER by-item }
```

where

target-item

is a field name, a numeric literal, the predefined aggregates SUM and COUNT, an arithmetic expression, an IF/THEN/ELSE expression, a user table name, a System Variable clause. The data type of the target-item used in a PCT clause must be numeric.

If *target-item* is an expression, that expression cannot contain an aggregate except a simple SUM or COUNT. If *target-item* is a user variable, that variable must not have been assigned an aggregate value unless the value is the result of a simple SUM or COUNT operation.

by-item

is the name of a numeric field grouped by a BY or BY DESC clause.

Using PCT OVER ALL

When the PCT OVER ALL clause is used, ENFORM prints a percentage of the grand total for the numeric *target-item* in place of each *target-item* value. When only PCT is specified, ENFORM assumes PCT OVER ALL.

Using PCT OVER By-item

When the PCT OVER *by-item* clause is used, ENFORM prints the percentage of the instances of the numeric *target-item* within the group in place of each *target-item* value. The group must have been defined earlier by a BY or BY DESC clause. In the following example, ENFORM prints the percentage value of the cost of one part over the total value of all the parts at each location. For example, consider the following query and report:

```
LIST BY location,
     partnum,
     price,
     price PCT OVER location;
```

LOCATION	Part Number	PRICE	PCT PRICE
H76	3102	4800.00	66.67
	7301	2400.06	33.33
H57	2402	7500.00	41.67
	3103	10500.00	58.33
...	...	...	...

Combining Percentages and Subtotals

The SUBTOTAL OVER clause can be combined with the PCT OVER clause, causing ENFORM to print the percentage the subtotal is of the total value of the *target-item*. ENFORM does not print the value of each *target-item*. The percentage values do not always exactly total 100% due to truncation during division. If you execute the following query for example, the total percentage values do not equal 100 %:

```
OPEN employee;
LIST BY regnum, empname, salary PCT, SUBTOTAL OVER regnum;
```

PCT Clause Used With User Variable

When a numeric *target-item* with a PCT clause is assigned to a user variable, ENFORM makes the assignment to the user variable first, before the percentage is calculated. When the user variable is referenced as a *target-item* in a LIST statement or as an element within a *target-item* of a LIST statement, ENFORM uses the value of the user variable. When the user variable is referenced in a print list, ENFORM uses the value of the percentage for the user variable.

Restrictions

Note that you cannot combine the PCT clause with the CUM clause.

SKIP CLAUSE

The SKIP clause allows you to indicate the number of lines ENFORM should move forward before continuing printing. The syntax of the SKIP clause is:

```
SKIP [ number ]

where

  number

    an integer representing the number of lines.
```

Sometimes it is desirable to print more than one target-item under the same column heading. If you want to do this, you can specify a SKIP clause after the first target-item, then use the TAB clause to position the printing of the second target-item under the first target-item. The second target-item should be modified by a NOHEAD clause to prevent ENFORM from printing the unwanted column heading. ENFORM prints the target-items under a a suppressed column heading, one target-item per line.

The option variable @VSPACE affects the number of lines ENFORM advances when the SKIP clause is specified. Refer to the Option Variable Clause in this section for more information.

Specifying the symbol / (slash) is equivalent to SKIP or SKIP 1.

SKIP clause With a LIST Target-item or By-item

The SKIP clause can precede or follow a target-item or by-item within a LIST statement. If *number* is specified, ENFORM moves forward the specified number of lines before printing continues. If *number* is not specified, ENFORM moves forward to the next line.

In the following example, address and city are printed in the same column on separate lines.

```
LIST address,SKIP,
     city NOHEAD SKIP 2;

ADDRESS
-------
UNIVERSITY WAY
PHILADELPHIA

100 CALIFORNIA STREET
SAN FRANCISCO
```

SKIP Clause With a Print List

The SKIP clause can be part of the print list of an AT END statement or clause, an AT START statement or clause, a FOOTING statement or clause, a SUBFOOTING statement or clause, a SUBTITLE statement or clause, or an AFTER CHANGE or BEFORE CHANGE clause. When *number* is specified, ENFORM moves forward the specified number of lines every time the SKIP clause is processed. When *number* is not specified, ENFORM moves forward to the next line when the SKIP clause is processed.

SPACE CLAUSE

The SPACE clause allows you to specify horizontal spacing. The syntax of the SPACE clause is:

```
SPACE [ number ]

where

   number

      is an integer.
```

SPACE Clause With a LIST Target-item or By-item

The SPACE clause can precede either a target-item or a by-item within a LIST statement. When *number* is specified, ENFORM inserts the specified number of spaces every time the SPACE clause is processed. When *number* is not specified, one space is inserted.

The default spacing between columns on a report is initially set to two spaces. It can be temporarily overriden by using the SPACE clause with the *number* parameter. To delete all spaces between columns, use SPACE 0.

SPACE Clause With a Print list

The SPACE clause can be part of a print list for an AFTER CHANGE, BEFORE CHANGE, AT START PRINT, AT END PRINT, FOOTING, SUBFOOTING, SUBTITLE, or TITLE clause or statement. When *number* is specified, ENFORM inserts the specified number of spaces every time the SPACE clause is processed. When *number* is not specified, one space is inserted.

SUBFOOTING CLAUSE

The SUBFOOTING clause allows you to specify printing of information at the bottom of each page preceding the footing for the current report. This clause is an optional part of the LIST statement. The syntax of the SUBFOOTING clause is:

```
SUBFOOTING print-list [ CENTER ]
```

where

print-list

contains any combination of literals, FORM, SKIP, SPACE, or TAB clauses. A *print-list* can also contain the following elements that can be modified by AS, AS DATE, or AS TIME clauses: field names, arithmetic expressions, IF/THEN/ELSE expressions, System Variable clauses, JULIAN-DATE clauses, TIMESTAMP-DATE clauses, TIMESTAMP-TIME clauses, user variables, or parameter names.

The clauses that can be part of a *print-list* are described in this section. The other elements are described in Section 3.

Specifying Field Names in a SUBFOOTING Clause

If you specify a field name within a *print-list* of a SUBFOOTING clause, ENFORM prints the same field value as the last row of data on the current page. A field name appearing within the SUBFOOTING clause need not be explicitly included within the LIST statement. If the field name, ENFORM effectively adds the field to the LIST statement with a NOPRINT clause.

Spacing Considerations

By default the footing begins in the same position as the leftmost report column. Using SPACE or TAB clauses as the first element of the *print-list* overrides this default. SPACE or TAB clauses can also appear anywhere within the *print-list*. In the following example, the SPACE clause causes the two literals to be separated by 15 spaces:

```
SUBFOOTING "Inventory" SPACE 15 "Parts in Stock",

Inventory               Parts in Stock
```

If you specify either a SKIP clause or the slash symbol (/) within a *print-list*, ENFORM advances one or more lines before printing the rest of the SUBFOOTING *print-list*. The number of lines advanced can be affected by one or more of the following: the digit (if any) following the keyword SKIP, the number of slashes specified, or the option variable @VSPACE. In the following example, the SKIP clause causes two lines to be printed:

```
SUBFOOTING "Report 2-A" SKIP "Total Sales",
```

The following subfooting prints on the current report:

```
Report 2-A
Total Sales
```

Using the FORM clause within a SUBFOOTING statement forces a new page. ENFORM continues printing the remainder of the SUBFOOTING *print-list*, starting at the top of the next physical page. The page number remains the same. A single logical page can span multiple physical pages, such that a TITLE can appear on one page, the data on the next, and a SUBFOOTING on the next. The same page number applies to all physical pages in the logical page.

Using the CENTER clause centers the subfooting within the leftmost and rightmost columns of the report.

The CENTER, Option Variables, SKIP, SPACE, and TAB clauses are described in this section.

Subfooting for Current Report or All Reports

A SUBFOOTING clause within a LIST statement prints a subfooting only for the current report. A SUBFOOTING clause temporarily overrides a session-wide SUBFOOTING statement. A SUBFOOTING statement prints a subfooting at the bottom of each page for all subsequent reports in the current session.

SUBTITLE CLAUSE

The SUBTITLE clause allows you to specify printing of information at the top of each page immediately following the title for the current report. This clause is an optional part of the LIST statement. See also the TITLE clause in this section and the SUBTITLE statement in Section 4. The syntax of the SUBTITLE clause is:

```
SUBTITLE print-list [ CENTER ]
```

where

print-list

contains any combination of literals, FORM, SKIP, SPACE, or TAB clauses. A *print-list* can also contain the following elements that can be modified by AS, AS DATE, or AS TIME clauses: field names, arithmetic expressions, IF/THEN/ELSE expressions, System Variable clauses, JULIAN-DATE clauses, TIMESTAMP-DATE clauses, TIMESTAMP-TIME clauses, user variables, or parameter names.

The clauses that can be part of a *print-list* are described in this section. The other elements are described in Section 3.

Specifying a Field Name in a SUBTITLE Clause

If you specify a field name within a *print-list* of a SUBTITLE clause, ENFORM prints the same field value as in the first row of the report. A field name appearing within the SUBTITLE clause need not be explicitly included within the associated LIST statement. If the field name is not included, ENFORM in effect adds it with a NOPRINT clause.

Spacing Considerations

By default the subtitle begins in the same position as the leftmost report column. Using SPACE or TAB clauses as the first element of the *print-list* overrides this default. SPACE or TAB clauses can also appear anywhere within the *print-list*. In the following example, the SPACE clause causes the two literal to be separated by 15 spaces:

```
SUBTITLE "Inventory" SPACE 15 "Parts in Stock",

Inventory               Parts in Stock
```

If you specify either a SKIP clause or the slash symbol (/) within a *print-list*, ENFORM advances one or more lines before printing the rest of the SUBTITLE *print-list*. The number of lines advanced can be affected by one or more of the following: the digit (if any) following the keyword SKIP, the number of slashes specified, or the option variable @VSPACE. In the following example, the SKIP clause causes two lines to be printed:

```
SUBTITLE "Report 2-A" SKIP "Total Sales",

Report 2-A
Total Sales
```

Using the FORM clause within the *print-list* causes ENFORM to start a new page. ENFORM continues with the remainder of the *print-list* starting at the top of the next physical page. The page number remains the same.

Using the CENTER clause centers the subtitle within the leftmost and rightmost columns of the report.

The CENTER, Option Variable, SKIP, SPACE, and TAB clauses are described in this section.

Subtitle for Current Report or All Reports

A SUBTITLE clause within a LIST statement prints a subtitle only for the current report. It temporarily overrides the session-wide SUBTITLE statement. A SUBTITLE statement prints a subtitle at the top of each page immediately following the title for all subsequent reports in the current session.

SUBTOTAL CLAUSE

The SUBTOTAL clause allows you to specify the printing of a subtotal for a numeric target-item. This clause is an optional part of the LIST statement. The syntax for the SUBTOTAL clause is:

```
target-item  SUBTOTAL [ OVER by-item ]
```

where

target-item

is a record name, a field name, a numeric literal, a predefined aggregate, a user aggregate, an arithmetic expression, an IF/THEN/ELSE expression, a user table name, a user variable, or a System Variable clause. The data type of the target-item being subtotaled must be numeric.

by-item

is the name of a field grouped by a BY or BY DESC clause.

When the SUBTOTAL clause is specified, ENFORM prints prints the subtotals for each *target-item* within a by-item value. ENFORM prints the subtotal in the column of the target-item being subtotaled and precedes the subtotal by a row of underline characters specified by the @UNDERLINE option variable. ENFORM marks the subtotal with a subtotal string specified by the @SUBTOTAL-LABEL option variable. Refer to the Option Variable clauses in this section.

When a SUBTOTAL OVER clause is used, ENFORM subtotals the *target-item* each time the value of the specified *by-item* changes. When more than one SUBTOTAL OVER clause is specified, ENFORM prints the subtotals in the order that the clauses are entered in the LIST statement.

When a SUBTOTAL clause is used without OVER, ENFORM prints a subtotal in the specified *target-item* column each time the value of any by-item appearing to the left of the *target-item* changes. ENFORM prints the subtotals using a minor to major order precedence; that is: ENFORM prints the subtotal for the first by-item to the left of the *target-item*, followed by the subtotal for the second by-item to the left of the *target-item*, and so on until the subtotal for the last by-item appearing in the report is printed.

SUBTOTAL does not generate a grand total at the end of the report. If a grand total is desired, use the TOTAL clause. The TOTAL clause is described in this section.

If the width of the value of a subtotal exceeds the width of the format specified for a *target-item*, an overflow condition occurs causing asterisks to be printed in place of the value. To prevent this, enlarge the *target-item* display format by adding an AS clause to the *target-item* being subtotaled. The AS clause is described in this section.

SUPPRESS CLAUSE

The SUPPRESS clause allows you to eliminate certain records from being printed in the report. The records still contribute to the report calculations. This clause is an optional part of the part of the LIST statement. This clause cannot be used with the FIND statement. The syntax of the SUPPRESS clause is:

```
SUPPRESS [ WHERE ] logical-expression
```

where

logical-expression

is an expression returning a true or false value. See Section 3 for more information.

The SUPPRESS clause defines a condition or conditions that prevents specific records from printing throughout a report. For example, in the following query, the SUPPRESS clause causes ENFORM to print only the inventory, part number, and part name where the inventory is greater than zero:

```
LIST ASCD inventory,partnum,partname,
  SUPPRESS WHERE inventory GT 0;
```

This query generates the following report:

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

      -100     2001   DECIMAL ARITH
       -32     6402   TERM CRT PAGE
       -16     6201   SYNC CONTROLLER
        -1     5504   LP 900 LPM
         0     5505   LP 1500 LPM
```

ENFORM still includes the suppressed records in AFTER CHANGE and BEGIN CHANGE clauses, subtotals, totals, and other calculations specified for the report. Note that the value of the first record of an AFTER CHANGE clause or the last record of a BEFORE CHANGE clause is used for the print list whether or not that record is printed.

Aggregates cannot be used in a SUPPRESS WHERE clause; however, you can reference a user variable that has been assigned the aggregate value.

SYSTEM VARIABLE CLAUSES

The System Variable clauses allow you to obtain the current value for the current date, time, line number, and page number. The syntax for the System Variable clauses is:

```
@DATE
@TIME
@LINENO
@PAGENO
```

Printing the Current Date or Time

The *@DATE* and *@TIME* System Variable clauses return the current date and time in internal format. When used in an expression, ENFORM treats them as numeric literals.

When the *@DATE* or *@TIME* system variables are to be printed on a report, convert them to a display format using the AS DATE or AS TIME clauses. Frequently the default format is satisfactory and can be specified by:

```
@DATE AS DATE *
```

For more information on date and time display formats, refer to to AS DATE and AS TIME clauses in this section.

Printing Line Numbers

The *@LINENO* System Variable clause prints the current line number within a page of the report. *@LINENO* can be a target-item in a LIST statement, part of an AFTER CHANGE, BEFORE CHANGE, AT START, AT END, FOOTING, TITLE, SUBFOOTING, and SUBTITLE clauses or statements.

The *@LINENO* option variable specifies how many lines are printed on a page. It is initially set to 60 lines. To change the number of lines to be printed per page, reset the *@LINENO* Option Variable clause. Refer to the SET statement in Section 4 and the Option Variable clauses in this section.

Printing Page Numbers

The *@PAGENO* System Variable prints the page number. The following example prints the page number at the top of a page:

```
TITLE TAB 100 "Page " @PAGENO;

                                                    Page 10
```

TAB CLAUSE

The TAB clause allows you to specify in which column in the report a target-item or by-item is to begin. This clause is an optional part of the LIST statement and must not be specified in the FIND statement. The syntax of the TAB clause is:

```
TAB [ number ]

where

  number

    is an integer.
```

The TAB clause specifies which column to tab to before printing an element on a report. Care must be taken not to overlap elements. If overlap occurs, the last numbers or characters specified appear on the printed report. Note that TAB never causes ENFORM to advance to the next line, so you can tab backwards on the current line.

TAB Clause With a LIST Target-item or By-item

The TAB clause can precede a target-item or by-item within a LIST statement. If *number* is specified, ENFORM tabs to that position before the next target-item or by-item is printed; if *number* is not specified, ENFORM begins printing in column one.

TAB Clause With a Print List

The TAB clause can be part of a print list for an AT END statement or clause, an AT START statement or clause, a FOOTING statement or clause, a SUBFOOTING statement or clause, a SUBTITLE statement or clause, and the AFTER CHANGE and BEFORE CHANGE clauses. When *number* is specified, ENFORM tabs to that position before the next element is printed on the current line. If *number* is not specified, ENFORM assumes column one.

TIMESTAMP-DATE CLAUSE

The TIMESTAMP-DATE clause extracts the date portion of a timestamp field that has been created by the GUARDIAN procedure TIMESTAMP. The syntax of the TIMESTAMP-DATE clause is:

```
TIMESTAMP-DATE ( field-name )
```

where

field-name

is the name of a field to which the TIMESTAMP-DATE clause returns a date value. The field must be described in the dictionary.

You must define the field that receives the date value from the TIMESTAMP-DATE clause in your data dictionary. You must define the field as a six-character alphanumeric field, such as:

```
05  TIME-STAMP    TYPE CHARACTER 6.
```

The value returned to this field is a quantity of days in internal format.

Refer to the TIMESTAMP-TIME clause in this section for extracting the time portion of a timestamp field.

TIMESTAMP-TIME CLAUSE

The TIMESTAMP-TIME clause extracts the time portion of a timestamp field that has been created by the GUARDIAN procedure TIMESTAMP. The syntax of the TIMESTAMP-TIME clause is:

```
TIMESTAMP-TIME ( field-name )
```

where

field-name

is the name of a field to which a time value is returned. The field must be described in your data dictionary.

Your data dictionary must contain a definition of the field to which the TIMESTAMP-TIME clause returns a time value. You must define the field as a six-character alphanumeric field, such as:

```
05  TIME-STAMP    TYPE CHARACTER 6.
```

The value returned to this field is a quantity in one hundredths of a second in internal time format. You can obtain this value in seconds by using the AS clause with an integer edit descriptor or by using the AS TIME clause. If you need to obtain this value in one hundredths of a second, you must write your own conversion routine.

Refer to the TIMESTAMP-DATE clause in this section for extracting the date portion of a timestamp field.

TITLE CLAUSE

The TITLE clause allows you to specify printing of information at the top of each page for the current report. See also the SUBTITLE clause in this section and the TITLE statement in Section 4. The syntax of the TITLE clause is:

```
TITLE print-list [ CENTER ],
```

where

print-list

contains any combination of literals, FORM, SKIP, SPACE, or TAB clauses. A *print-list* can also contain the following elements that can be modified by AS, AS DATE, or AS TIME clauses: field names, arithmetic expressions, IF/THEN/ELSE expressions, System Variable clauses, JULIAN-DATE clauses, TIMESTAMP-DATE clauses, TIMESTAMP-TIME clauses, user variables, or parameter names.

The clauses that can be part of a *print-list* are described in this section. The other elements are described in Section 3.

Specifying Field Names in a TITLE Clause

If you specify a field name within a *print-list* of a TITLE clause, ENFORM prints the same field value as in the first row of the page. A field name appearing within the TITLE clause need not be explicitly included within the associated LIST statement. If field name is not included, ENFORM in effect adds it with a NOPRINT clause.

Spacing Considerations

By default the title begins in the same position as the leftmost report column. Using SPACE or TAB clauses as the first element of the *print-list* overrides this default. SPACE or TAB clauses can also appear anywhere within the *print-list*. In the following example, the SPACE clauses causes the two literals to be separated by 15 spaces:

```
TITLE "Inventory" SPACE 15 "Parts in Stock",

Inventory               Parts in Stock
```

If you specify either a SKIP clause or the slash symbol (/) within a *print-list*, ENFORM advances one or more lines before printing the rest of the AFTER CHANGE *print-list*. The number of lines advanced can be affected by one or more of the following: the digit (if any) following the keyword SKIP, the number of slashes specified, or the option variable @VSPACE. In the following example, the SPACE clause causes two lines to be printed:

```
TITLE "Report 2-A" SKIP "Total Sales",

Report 2-A
Total Sales
```

Using the FORM clause within the *print-list* causes ENFORM to start a new page and continue with the rest of the *print-list*. The page number remains the same. A single logical page can span multiple physical pages such that a TITLE appears on one page, the data on the next, and a FOOTING on the next. The same page number applies to all physical pages in a logical page.

Using the CENTER clause centers the title within the leftmost and rightmost columns of the report.

The CENTER, Option Variables, SKIP, SPACE, and TAB clauses are described in this section.

Title for Current Report or All Reports

A TITLE clause within a LIST statement prints a title only for the current report. It temporarily overrides the session-wide TITLE statement. A TITLE statement prints a title at the top of each page for all subsequent reports in the current session.

Overriding Session-Wide Title

Temporarily override a session-wide TITLE statement by using the TITLE clause with " " for the *print-list* parameter.

TOTAL CLAUSE

The TOTAL clause prints the grand total for a numeric target-item. This clause is an optional part of the LIST statement and cannot be specified in a FIND statement. The syntax of the TOTAL clause is:

```
{target-item} TOTAL
{by-item     }
```

where

`target-item`

is a record name, a field name, a numeric literal, a predefined aggregate, a user aggregate, an arithmetic expression, an IF/THEN/ELSE expression, a user table name, a user variable, a System Variable clause. The data type of the *target-item* being totaled must be numeric.

`by-item`

is a field whose values are grouped by a BY or BY DESC clause.

When the TOTAL clause is specified, ENFORM prints the grand total for a *target-item* or *by-item* after printing the last value of the *target-item* or *by-item*. ENFORM precedes the total by two rows of underline characters, as specified by the @UNDERLINE option variable. The Option Variable clauses are discussed in this section.

If the width of the value of a total exceeds the width of the *target-item* or *by-item* display format, an overflow condition occurs causing asterisks to be printed in place of the total. To prevent this, enlarge the display format by adding an AS clause to the *target-item* being totaled. The AS clause is described in this section.

WHERE CLAUSE

The WHERE clause allows you to qualify the records that contribute to the target-record. The syntax of the WHERE clause is:

```
WHERE logical-expression
```

where

logical-expression

is a condition that returns a true or false value. See Section 3 for more information about logical expressions.

The logical expression in a WHERE clause defines which records are restricted from contributing to the target-record. For example:

```
WHERE empname BEGINS WITH "BROWN",
```

Using the WHERE Clause to Specify a Link

Using the WHERE clause to specify a link causes only the records which satisfy the condition specified in the logical expression to be included in the report. The link created by the WHERE clause remains in effect only for the associated LIST or FIND statement. For a more complete description of linking, refer to the LINK statement in Section 4. The following example shows how the WHERE clause is used to specify a link:

```
WHERE parts.partnum EQ odetail.partnum,
```

SECTION 6

COMMANDS

This section contains a complete description of the syntax of the ENFORM commands. The commands are arranged in alphabetical order.

ENFORM commands are compiler directives that instruct the ENFORM compiler/report writer to perform a specific operation. The compiler/report writer recognizes a command by the presence of a question mark in column 1. You can either enter commands at the level of the ENFORM prompt (>) or place commands in an EDIT file. ENFORM executes commands in an EDIT file as if they were entered interactively with one exception: ENFORM ignores a ?RUN command in an EDIT file.

The following commands are not saved in a compiled query file (created by the ?COMPILE command): ?ATTACH, ?COMPILE, ?EDIT, ?EXECUTE, ?EXIT, ?OUT, ?RUN, ?SECTION, ?SHOW, ?SOURCE. For this reason, these commands are not used:

- Through the host language interface (described in the *ENFORM Users Guide*). The host language interface uses a compiled query file.
- When you specify the ?EXECUTE command. The ?EXECUTE command can only be used with a compiled query file.
- If the filename specified in the IN option of the ENFORM command is a compiled query file.

Table 6-1 shows the ENFORM commands and their functions.

Table 6-1. Summary of Commands

Command	Function
?ASSIGN	associates a new physical file with a record description.
?ATTACH	specifies a query processor (QP) to use during an ENFORM session.
?COMPILE	compiles an ENFORM query and saves it in a compiled query file.
?DICTIONARY	names a subvolume which contains a dictionary. It also clears the internal table and reclaims table space.
?EDIT	accesses the Tandem Text Editor without leaving ENFORM.
?EXECUTE	executes a compiled query file.
?EXIT	terminates the current ENFORM session.
?HELP	displays information about the syntax of the ENFORM language. This command can also display user-defined information.
?OUT	specifies the output device for a report.
?RUN	compiles and executes ENFORM source queries in an EDIT file.
?SECTION	identifies a section of ENFORM commands and statements within an EDIT file.
?SHOW	displays information about the environment of the current ENFORM session.
?SOURCE	reads an EDIT file or a collection of commands and statements within an EDIT file.

?ASSIGN COMMAND

The ?ASSIGN command associates a physical file with a dictionary record description or a generic file. This command is functionally equivalent to the Command Interpreter ASSIGN command. The syntax of the ?ASSIGN command is:

```
?ASSIGN [ {record-name       } [ , ] physical-filename ]
        [ {generic-file-name } [TO]                    ]

  [ , create-open-spec  ... ]

?ASSIGN  record-name , , create-open-spec  ...
```

where

record-name

is the name of a record description from a DDL dictionary.

generic-filename

is the name of one of the ENFORM generic files described in Section 2. A generic file is used to store some class of ENFORM output. The following names are allowed:

QUERY-COMPILER-LISTING
QUERY-REPORT-LISTING
QUERY-STATISTICS
QUERY-STATUS-MESSAGES
QUERY-WORK-AREA
QUERY-SORT-AREA
QUERY-QPSTATISTICS
QUERY-QPSTATUS-MESSAGES.

physical-filename

For *record-name*, *physical-filename* is the name of a physical file being associated with the record description. TO must be specified if the file name is TO. A partial file name is expanded using the default system, volume, and subvolume names.

For *generic-file-name*, *physical-filename* is the name of the physical device to which you want ENFORM to direct output. →

```
create-open-spec
```

is one file creation or open attribute. Only exclusion spec is used by ENFORM. *create-open-spec* is of the form:

```
{ extent-spec       }
{ CODE file-code    }
{ exclusion-spec    }
{ access-spec       }
{ REC record-size   }
{ BLOCK block-size  }
```

where

```
extent-spec
```

is one of the following:

```
{EXT [ ( ] pri-extent-size [ ) ]                }
{EXT ( [ pri-extent-size ] , sec-extent-size )  }
```

```
exclusion-spec
```

is one of the following:

```
{ EXCLUSIVE }
{ SHARED    }
{ PROTECTED }
```

```
access-spec
```

is one of the following:

```
{ I-O    }
{ INPUT  }
{ OUTPUT }
```

The ?ASSIGN command can be entered at any time prior to running a query. Each time an ?ASSIGN command is entered, an entry of the assignment is made in the internal table. The internal table holds up to 32 assignments. When the internal table is full, the ?ASSIGN command can be entered without any parameters to clear all of the assignments.

Each DDL dictionary RECORD statement describes records in a physical file, named by a DDL FILE IS clause. ENFORM uses the physical file name in the DDL FILE IS clause to locate the records and retrieve information. There might be times when the actual records desired are not in the physical file named in the dictionary. An example would be when test records are used in place of actual records. The ?ASSIGN command temporarily overrides the physical file named by the DDL FILE IS clause. In the following example, the employee record description is assigned to the physical file $data.database.sample:

```
?ASSIGN employee TO $data.database.sample
```

The ?ASSIGN command can be used to assign the generic ENFORM files to a physical device. In the following example, the generic file QUERY-COMPILER-LISTING is assigned to a printing device named $ep:

```
?ASSIGN QUERY-COMPILER-LISTING TO $s.#ep
```

The ?ASSIGN command can be used to change the exclusion specification for a file; the default is SHARED. If the DDL physical file name is not being overridden, only the record name and the exclusion specification need be specified. For example:

```
?ASSIGN employee,,SHARED
```

When ENFORM is run interactively, a ?ASSIGN command overrides a Command Interpreter ASSIGN command entered before the ENFORM command. When ENFORM is run with a command file providing the input, however, a Command Interpreter ASSIGN command entered before the ENFORM command overrides a ?ASSIGN command in the command file.

?ATTACH COMMAND

The ?ATTACH command allows you to specify a query processor to use during an ENFORM session. The syntax of the ?ATTACH command is:

```
?ATTACH [ process-name ]

where

  process-name

    is the name of a query processor.
```

When one or more server query processors exist on a Tandem system, you can use the ?ATTACH command to specify the query processor you want to use. If you do not specify the ?ATTACH command, ENFORM defaults to a dedicated query processor. A dedicated query processor is an ENFORM process that is created for and provides services to an individual compiler/report writer process. The dedicated query processor runs for the duration of an ENFORM session until either an error occurs or an ?ATTACH command is issued specifying a new query processor. When you enter the ?ATTACH command without a process name, ENFORM starts a new dedicated query processor for your use.

An ?ATTACH failure is not evident until the first FIND or LIST statement is executed. The possible ?ATTACH failures and the actions you can take are as follows:

- The server query processor already has the maximum number of users it can handle. In this case, you can wait and then try again specifying the same query processor; name a different server query processor; or select a dedicated query processor.

- The server query processor does not exist. In this case, you can start the server query processor, name a server query processor that does exist, or select a dedicated query processor.

- The attached server query processor reached one of two limits set for that particular server query processor. Either the query reached a data base read limit or the cost strategy specified by ENFORM exceeded its limit. In this case, you can: name a server query processor with greater limits or without built-in limits; change the query; or select a dedicated query processor. The @READS and @COST-TOLERANCE Option Variable clauses are discussed in Section 5.

?COMPILE COMMAND

The ?COMPILE command compiles a query and stores it in a physical file. The syntax of the ?COMPILE command is:

```
?COMPILE edit-filename [ ( section-name, ...) ] TO compiled-physical-filename
```

where

edit-filename

is the name of the EDIT file containing the ENFORM query.

(section-name, ...)

is the name of the section(s) within an EDIT file. The list must be enclosed within parentheses.

compiled-physical-filename

is the name of the physical file to contain the compiled query.

The ?COMPILE command compiles a query. The ENFORM internal table becomes part of the compiled query file. If *physical-filename* existed before compilation, the old version is purged and a new physical file is created. If errors are produced during a compilation attempt, the original compiled query file is not affected at all.

The ?COMPILE command only compiles one LIST or FIND statement. If your EDIT file contains more than one LIST or FIND statement, ENFORM compiles the first LIST or FIND statement to the compiled query file. ENFORM does not compile any statements or commands following the first LIST or FIND statement; instead ENFORM continues processing, executing the statements and commands as they are encountered in the EDIT file.

When the LIST or FIND statement of the query is written to the compiled query file, most of the commands, options, and other statements that provide environmental information are stored in the internal table. The internal table is then saved in the compiled query file.

ENFORM assigns a file code of 888 to the compiled query file. ENFORM creates this code to identify the physical file as a stored query produced by ENFORM.

A query called by a host language program must be compiled first. Other queries do not need to be compiled before run time; however, there are some advantages to creating a compiled query file. The compiled query file protects the query from being inadvertently changed. Less processing time is required when the compiled query file is saved and reused for runs.

?DICTIONARY COMMAND

The ?DICTIONARY command names a subvolume that contains your dictionary. It also clears the internal table and reclaims table space. The ?DICTIONARY command is the same as the DICTIONARY statement.

```
?DICTIONARY [ dict-subvol-name ]

where

  dict-subvol-name

    is the name of the subvolume where the dictionary files reside. Refer to the GUARDIAN
    Operating System Programming Manual, for information about specifying file names.
```

The dictionary identified in a ?DICTIONARY command must be created by the Data Definition Language compiler. Release T9102C10 of ENFORM accepts only dictionaries produced by DDL version T9100D00 or later. ENFORM issues an error message if an attempt is made to use a dictionary compiled with an earlier version of DDL. These dictionaries must be recompiled with DDL version T9100D00 or later before ENFORM is used. Refer to the *Data Definition Language (DDL) Programming Manual*, for complete instructions.

Identifying the Dictionary

There are two ways to indicate where the dictionary resides.

- The volume and subvolume where the dictionary resides can be specified as a part of the Command Interpreter ENFORM command. If none is specified, it is assumed that the dictionary resides on the default volume and subvolume.

- The DICTIONARY statement or ?DICTIONARY command can specify where the dictionary resides. Either overrides the dictionary identified at the time of the Command Interpreter ENFORM command. When a new dictionary is specified, the old internal table is cleared.

Clearing Internal Tables

Entering the ?DICTIONARY command without a volume and subvolume name is a simple means of clearing the entire internal table, without changing the dictionary. To clear only certain items of the internal table, refer to the CLOSE and DELINK statements in Section 4. The items cleared by the ?DICTIONARY command are:

- All record descriptions from previous OPEN statements
- All previous linking relationships
- All user variable, user aggregate, and user table definitions

  All parameter definitions.

?EDIT COMMAND

The ?EDIT command allows you to access the Editor without leaving ENFORM. The syntax of the ?EDIT command is:

```
?EDIT [ edit-filename ]

where

   edit-filename

      is the name of an EDIT file.
```

A query can be stored in an EDIT file. The EDIT file can be created or changed without leaving ENFORM. An ?EDIT command invokes the Editor and operates just as if the Command Interpreter EDIT Command was used. If a new filename is specified while using the Editor, that name becomes the default *edit-filename* for use in a subsequent ?EDIT or ?RUN command. To exit the Editor, enter EXIT at the EDIT prompt (*) and control returns to ENFORM.

?EXECUTE COMMAND

The ?EXECUTE command executes a compiled query file. The syntax of the ?EXECUTE command is:

```
?EXECUTE compiled-physical-filename

where

   compiled-physical-filename

      is the name of the physical file containing the stored compiled query.
```

The ?EXECUTE command resets the internal table to the same state that existed at time of compilation. The ?EXECUTE command changes the dictionary being used for the duration of the execution. Internal table information from the compiled query file is used until execution terminates, then ENFORM uses the dictionary specified before you entered the ?EXECUTE command.

If the stored compiled query requires a parameter value passed to it during execution, the Command Interpreter PARAM command can be issued prior to executing the query. For information on the PARAM command, refer to the *GUARDIAN Operating System Command Language and Utilities Manual.*

The ?EXECUTE command only accepts a physical file with a file code of 888. ENFORM assigns a file code of 888 when it creates a compiled query file.

?EXIT COMMAND

The ?EXIT command terminates the current ENFORM session. The syntax of the ?EXIT command is:

```
?EXIT
```

The ?EXIT command returns control to the Command Interpreter. It is the same as the EXIT statement. An alternate way of exiting ENFORM is to press the terminal CTRL and Y keys simultaneously.

?HELP COMMAND

The ?HELP command displays information about the syntax of the ENFORM language. The ?HELP command can also display information about user-defined topics if your system manager has provided this information. The syntax of the ?HELP command is:

```
?HELP [ help-element ]
```

where

help-element

is one of the topics for which help is available. Among these topics is the syntax of the ENFORM statements, clauses, commands, and language elements. To obtain a current list of the *help-elements*, enter the ?HELP command without *help-element*. ENFORM responds by displaying a list of all the topics for which help is available.

When specifying the ?HELP command, enter *help-element* in either upper or lower case characters. ENFORM accepts unambiguous abbreviations for *help-element*. If you use an ambiguous abbreviation, ENFORM displays help text for all *help-elements* beginning with those characters.

A *help-element* might be an ENFORM keyword (for example, LIST). If *help-element* is an ENFORM keyword, ENFORM displays the syntax of the statement, clause, command, or language element associated with that keyword. Alternatively, a *help-element* might be a category of ENFORM keyword (for example, report format). If *help-element* is a category, ENFORM displays the syntax for all ENFORM keywords in that category.

ENFORM retrieves the text of help messages from the ENFORM message table. By modifying the message table, you can modify the help text. Refer to the *ENFORM Users Guide* for information about modifying the message table.

The following example shows a list of topics that might be displayed when you enter the ?HELP command without *help-element*:

```
>?HELP
Help is available for the following topics:

STATEMENTS:
   AT END        AT START     CLOSE      DECLARE      DELINK
   DICTIONARY    EXIT         FIND       FOOTING      LINK
   LIST          OPEN         PARAM      SET          SUBFOOTING
   SUBTITLE      TITLE

CLAUSES:
  global modifier          modifier
  positional control       report format
  target-item

OTHER:
  aggregate                arithmetic expression
  logical expression       option variable
  print list               system variable

COMMANDS:
  ?ASSIGN      ?ATTACH      ?COMPILE     ?DICTIONARY    ?EDIT
  ?EXECUTE     ?HELP        ?OUT         ?RUN           ?SECTION
  ?SHOW        ?SOURCE
```

In the following example, ENFORM displays information about a target item when *?HELP target item* is entered:

```
>?HELP target item
A TARGET ITEM is a record name, a field name, a literal,
a user variable name, a parameter name, a system variable
clause, an aggregate, or an arithmetic expression.

Help is also available for SYSTEM VARIABLE, AGGREGATE, and
ARITHMETIC EXPRESSION.
```

?OUT COMMAND

The ?OUT command specifies the output device for a report. The syntax of the ?OUT command is:

```
?OUT [ physical-filename ]

where

  physical-filename

    is the name of the output device to which any subsequent report is directed.
```

The ?OUT command is used to name the physical device on which a report is printed. The physical device can be a printing device such as such as $mlp. The physical device can also be a file name. The ?OUT command remains in effect until a new ?OUT command is issued or the session is terminated.

Issuing the ?OUT command without specifying a *physical-filename* causes the report to be printed on the listing file specified by ENFORM's run-time OUT option.

The ?OUT command only affects where the report is sent. The source listing goes either to the default listing file specified by the OUT option of the ENFORM command or to the QUERY-REPORT-LISTING file.

The ?OUT command can be part of the EDIT file or precede an ?EXECUTE or ?RUN command. The ?OUT command is not saved in a compiled query file.

?RUN COMMAND

The ?RUN command executes a query or queries in an EDIT file. The syntax of the ?RUN command is:

```
?RUN [ edit-filename [ ( section-name, ...) ] ]
```

where

edit-filename

is the name of the EDIT file containing the ENFORM source query.

(section-name, ...)

is the name of one or more sections of the EDIT file. They must be enclosed within parentheses.

If both *edit-filename* and *section-name* are specified, that specific collection of commands and statements within the EDIT file identified by the section name is executed. Multiple sections of an EDIT file can be combined to create complete queries.

If only *edit-filename* is specified, ENFORM compiles and executes the entire EDIT file. If neither *edit-filename* nor *section-name* is specified, ENFORM compiles and executes the EDIT file specified by the most recent ?EDIT or ?RUN command.

Note that ENFORM issues a warning message and ignores a ?RUN command if:

- The ?RUN command appears in the file specified as the IN option of the ENFORM command.
- The ?RUN command appears in an EDIT file compiled and executed by a ?SOURCE command.
- The ?RUN command appears in an EDIT file compiled and executed by another ?RUN command.

?SECTION COMMAND

The ?SECTION command names a collection of ENFORM commands and statements within an EDIT file. The syntax of the ?SECTION command is:

```
?SECTION section-name

where

  section-name

    is the name to be used for a collection of ENFORM commands and/or statements within
    an EDIT file.
```

The ?SECTION command names a collection of ENFORM commands and/or statements within an EDIT file. The ?COMPILE, ?RUN, and ?SOURCE commands can be used to read a section or sections of an EDIT file.

The names for a section must follow these rules:

- must be unique
- must start with an alphabetic character or circumflex (^)
- can contain from 1-31 characters (names longer than 31 characters are truncated and a warning message is issued.
- can include numbers, hyphen (-) and/or circumflex (^)
- cannot contain embedded blanks
- cannot end with a hyphen (-).

?SHOW COMMAND

The ?SHOW command displays information about the environment of the current ENFORM session. The syntax of the ?SHOW command is:

```
?SHOW [ OPEN                    ]
      | LINK                    |
      | CONTROL                 |
      | LIMITS                  |
      | ASSIGN [ record-name ]  |
      | user-variable-name      |
      | record-name             |
      [ param-name              ]
```

where

user-variable-name

is the name of a user variable.

record-name

is the name of an opened dictionary record description.

param-name

is the name of a parameter.

Table 6-2 shows the environmental information displayed by the ?SHOW command.

Table 6-2. Environment Information Displayed by ?SHOW Command

Command	Display Message
?SHOW	Lists the various items that can be displayed with the ?SHOW command.
?SHOW OPEN	Lists the open record descriptions. If none have been opened, nothing is displayed. An OPEN statement does not actually open a physical file, but accesses the record description in the dictionary.
?SHOW LINK	Lists links that are in effect. If no record descriptions have been linked, nothing is displayed. A LINK statement does not actually link physical files, but accesses their record descriptions from the internal table.
?SHOW CONTROL	Displays the current values of all of the option variables. Option variables can be changed by setting the Option Variable clauses. Refer to the SET statement in Section 4 and the Option Variable clauses in Section 5.
?SHOW LIMITS	Displays the current space available for the symbol table, literals and formats, LINK table, PRINT table, PARAM table, and OVER clause table.
?SHOW ASSIGN	Displays all of the opened record descriptions and physical file name pairs specified by a ?ASSIGN command or Command Interpreter ASSIGN command. If none were assigned, nothing is displayed.
?SHOW ASSIGN record-name	Displays all ASSIGN table information related to the specified record name.
?SHOW user-var-name	Displays the current value(s) of the user variable or table.
?SHOW record-name	Displays the following for the specified opened record description. • the physical filename for the opened file description • the length of the file • each field's name, data type, length offset, number of occurrences, length of each occurrence, and whether a field is a key field or not.
?SHOW param-name	Displays the current value of the specified parameter.

?SOURCE COMMAND

The ?SOURCE command reads an EDIT file or a collection of commands and statements within an EDIT file. The syntax of the ?SOURCE command is:

```
?SOURCE edit-filename [ ( section-name, ...) ]
```

where

edit-filename

is the name of the EDIT file containing an ENFORM query.

(section-name, ...)

the name of the sections of the EDIT file. *section-name* must be enclosed with parentheses.

The ?SOURCE command can be used to read in an EDIT file or a collection of commands and/or statements within an EDIT file. When a ?SOURCE command is entered, the specified commands and/or statements are read in just as if they had been entered a line at a time. ?SOURCE commands can be nested up to a depth of four.

If both the *edit-filename* and *section-name* are specified, that specific collection of commands and statements of the EDIT file is read in. If only the *edit-filename* is specified, the entire EDIT file is read in.

Issuing the ?SOURCE command does not set the default EDIT file name for subsequent ?RUN or ?EDIT commands.

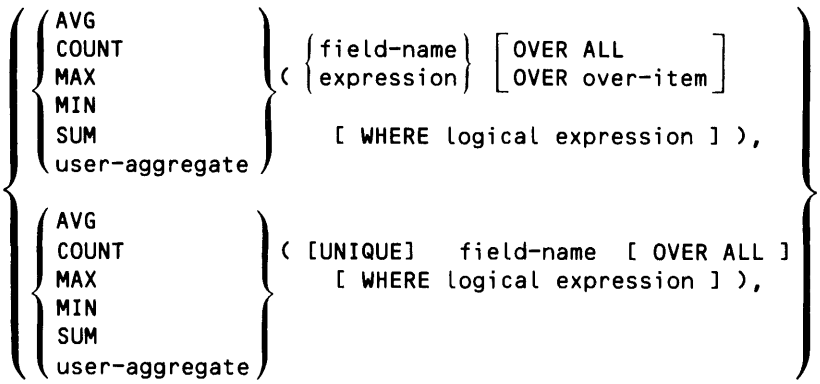
APPENDIX A

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:



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:

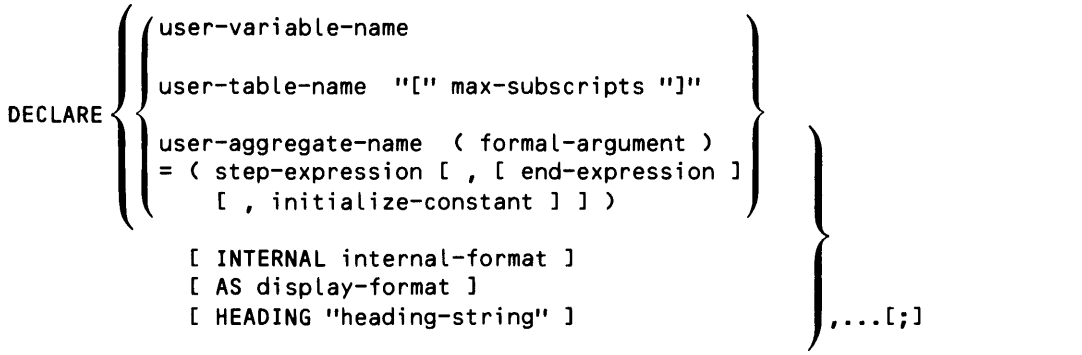
```
                { { BEGINS WITH }                  string-literal   }
                { {   ']'      }                                   }
                { { CONTAINS   }                                   }
                { {   '>'      }                                   }
field-name      { { conditional operator }                         }
                {                                                  }
                {   [NOT] { EQUAL }                                }
                {         { EQ    }          { value-range      } }
                {         { IS    }          { "["pattern-match"]" } }
                {         { =     }                              }
                {         { NE    }                              }
                {         { <>    }                              }

{ variable   }
{ field-name } [NOT] conditional operator { variable   }
{ expression }                            { field-name }
                                          { expression }
```

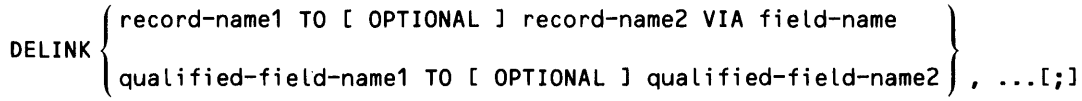
STATEMENTS

```
AT END [ PRINT print-list [ CENTER ] ] [ ; ]
```

```
AT START [ PRINT print-list [ CENTER ] ] [ ; ]
```

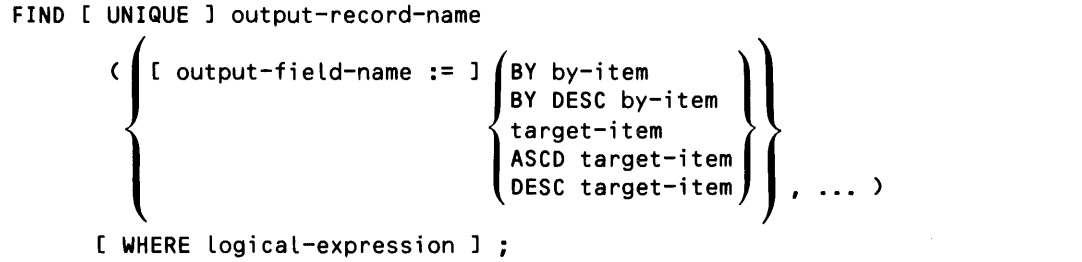
```
        { record-name      }
        { user-variable-name }
CLOSE   { user-aggregate-name } , ...[ ; ]
        { user-table-name  }
        { param-name       }
```


DECLARE
user-variable-name
user-table-name "[" max-subscripts "]"
user-aggregate-name (formal-argument)
= (step-expression [, [end-expression]
[, initialize-constant]])
[INTERNAL internal-format]
[AS display-format]
[HEADING "heading-string"]
,...[;]

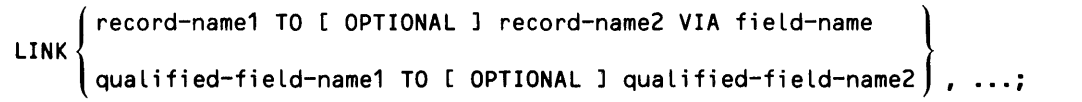

DELINK
record-name1 TO [OPTIONAL] record-name2 VIA field-name
qualified-field-name1 TO [OPTIONAL] qualified-field-name2
, ...[;]


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
, ...;

```
LIST [ UNIQUE ] { { BY by-item                          }
                  { BY DESC by-item                     }
                  { target-item                         }
                  { ASCD target-item                    }
                  { DESC target-item                    }
                  { user-var-name := target-item        }

                  [ CUM [ OVER ALL ]                    ]
                  [ CUM OVER by-item                    ]
                  [ PCT [ OVER ALL ]                    ]
                  [ PCT OVER by-item                    ]
                  [ TOTAL                               ]
                  [ SUBTOTAL                            ]
                  [ SUBTOTAL OVER by-item               ]
                  [ NOHEAD                              ]
                  [ NOPRINT                             ]
                  [ CENTER                              ]
                  [ HEADING "heading-string"            ]
                  [ AS display-format                   ]
                  [ AS DATE display-format              ]
                  [ AS TIME display-format               ], ...

                  [ FORM [ n ]                          ]
                  [ SKIP [ n ]                          ]
                  [ SPACE [ n ]                         ]
                  [ TAB [ n ]                            ], ...
                                                          }, ...

  [ 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, 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 messages 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 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 an 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

Record-name.KEY or KEY OF record-name is not allowed in a LINK statement or in an Aggregate clause.

!!! 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 was misspelled.

!!! ERROR [65] An aggregate may not be used in a print-list clause.

Self-explanatory.

!!! ERROR [66] Only one OPTIONAL LINK request allowed per LIST or FIND statement

Only one link (whether it is OPTIONAL or not) can be used when OPTIONAL is specified.

!!! 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

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 this 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 qualification relating it to any
     other record
```

Missing LINK statement or WHERE clause to link file descriptions.

```
!!!  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 an 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 numeric items.
```

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 a MEM (refer to the *GUARDIAN Operating System Programming Manual*) greater than 52.

```
***  WARNING  [111] The following LINKs were previously ignored; they are now
     being used:
       QUALIFIED-FIELD-NAME-1 is linked to QUALIFIED-FIELD-NAME-2 ...
```

ENFORM now uses all existing LINK statements. To obtain the results this query produced prior to ENFORM release T9102C09, remove all the LINK statements listed in WARNING [111]. Refer to the LINK statement discussion in Section 4.

```
!!!  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 [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 Section 5 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 Section 4 for more details.

```
!!!  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 Reference Manual.*

*** 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 management 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 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.

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, Volume 2.* 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, Volume 2.* xxxxxx are values in the hardware registers.

BUILDMK ERROR MESSAGES

*** 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 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 Primary key for key-sequenced file must be at offset 0 and have length 34

Self-explanatory.

*** ERROR Edit file contains ?HELP section but no ?MESSAGES section

The Edit file version of the message table must contain a ?MESSAGE section if a ?HELP section is included.

*** 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 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 not end with a hyphen

An identifier specified in the Edit file version of the message table must not end with a hyphen.

*** ERROR ?HELP subsection must contain at least one subsection

Self-explanatory.

*** ERROR ?MESSAGES section must contain at least one line of text

Self-explanatory.

*** ERROR ?VOCABULARY section must redefine at least one reserved word

Self-explanatory.

*** ERROR Identifier must contain less than 32 characters.

Self-explanatory.

*** 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 phrase that identifies a HELP section or subsection.

*** 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 Invalid edit file name was specified

An invalid Edit file name was specified in the BUILDMK command. Correct the Edit file name and re-issue 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 re-issue the BUILDMK command.

*** ERROR ?VOCABULARY must appear in columns 1-11 and all other columns must be blank

The characters ?VOCABULARY must appear in columns 1 through 11 of the first line of the ?VOCABULARY section. All other columns on this line must contain blanks.

*** ERROR ?MESSAGES must appear in columns 1-9 a nd all other columns must be blank

The characters ?MESSAGES must appear in columns 1 through 9 of the first line of the ?MESSAGES section. All other columns on this line must contain blanks.

*** ERROR ?HELP must appear in columns 1 thru 5 and all other columns must be blank

The characters ?HELP must appear in columns 1 through 5 of the first line of the ?HELP section. All other columns on this line must contain blanks.

*** 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 Key-sequenced file name parameter is missing

The name of the key-sequenced file is missing from the BUILDMK command. Re-issue the BUILDMK command and include the name of the key-sequenced file.

*** 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 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.

*** 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 file error occurred on *name* during the execution of BUILDMK.

*** FILE ERROR (POSITION failure) #file-error-number on name

The indicated file error occurred on *name* during the execution of BUILDMK.

*** FILE ERROR (READ failure) #file-error-number on name

The indicated file error occurred on *name* during the execution of BUILDMK.

*** FILE ERROR (WRITE failure) #file-error-number on name

The indicated file error occurred on *name* during the execution of BUILDMK.

APPENDIX C

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.

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 Description 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 a 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.

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, or relative files.

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 thru 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.volume.subvol.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 which 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 which 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 - the dictionary description of a record, including the record name, record type, field names, and data types, and key definitions.

Record Name - name given a record description in a DDL RECORD statement.

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.

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 specified by an ?ATTACH command, initiated separately from the 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, or 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 each field position in a record.

User Aggregate - a user declared aggregate. See Aggregate.

User Variable - a user declared element that can be used to store numeric or string literals, field values, and 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™ Reference Manual
82348 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—Technical Communications

TAPE

TAPE