SAS Publishing

Technical Report: S2-110 QUEUE Language for SYSTEM 2000® Software

Release 11.6 under IBM OS and CMS

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 1989. *Technical Report: S2-110 QUEUE Language for SYSTEM 2000® Software, Release 11.6 under IBM OS and CMS*. Cary, NC: SAS Institute Inc.

Technical Report: S2-110 QUEUE Language for SYSTEM 2000® Software, Release 11.6 under IBM OS and CMS

ISBN 1-55544-172-6

SAS Institute Inc., SAS Campus Drive, Cary, North Carolina 27513.

1st printing, 1989
2nd printing, December 1997

Note that text corrections may have been made at each printing.

SAS Publishing provides a complete selection of books and electronic products to help customers use SAS software to its fullest potential. For more information about our e-books, e-learning products, CDs, and hard-copy books, visit the SAS Publishing Web site at support.sas.com/pubs or call 1-800-727-3228.

Contents

Illustrations

Preface

SYSTEM 2000 software is a product of SAS Institute Inc. The software runs on CDC, IBM, and Unisys systems. This document is for users of IBM mainframes (series 370 or higher and equivalents) operating under OS and CMS environments.

This technical report is to be used as a reference manual for the QUEUE language. You must be familiar with the QUEST language. Any information you need about the QUEST language can be found in *SYSTEM 2000 QUEST Language, Release 11.6 under IBM OS and CMS*. Also, refer to that manual for information on CALC, UNIQUE, *NOW*, and *TODAY* commands, and usage considerations.

Syntax Notation

UPPERCASE words are keywords. They must be spelled exactly as given in the syntax. Abbreviations for keywords are also uppercase. If you have more than one choice, the choices are stacked vertically with a bar to the left. Select only one. A keyword without brackets is a required keyword.

Italicized words are generic terms representing words or symbols that you supply. If the term is singular, supply one instance of the term. If the term is plural, you can supply a list of terms, separated by commas. Each instance is followed by a comma except the last (or only) instance. If you have more than one choice, the choices are stacked vertically with a bar to the left. Select only one. A term without brackets is a required term.

[ ] Brackets enclose an optional portion of syntax. Multiple choices are stacked vertically with a bar to the left. Select only one. The brackets are not part of the command.

| A vertical bar to the left of vertically stacked choices means that you select one (which is either a required choice or an optional choice). You then continue to the next portion of syntax shown on the top line of the command syntax.

... Horizontal ellipsis points immediately following a required or optional portion of syntax indicate that the previous syntax can be repeated. In examples, either vertical or horizontal ellipses points indicate an omitted portion of output or a command.

b̸ The letter b with a slash indicates a significant blank in the syntax or in the output of a command. Generally, spaces indicate blanks, and the b̸ is used only for emphasis.

* The asterisk is the default system separator throughout this manual.

END The word END is the default entry terminator word throughout this manual.

| A vertical bar in the margin indicates a technical change or deletion in the text since the last printing.

Note: symbols shown within the syntax (such as parentheses, asterisks, commas) are required unless enclosed in brackets or specifically noted as optional.

Chapter 1

Using the QUEUE Language: Concepts

INTRODUCTION

The QUEST and QUEUE languages, subsets of the Self-Contained Language, can be used in either batch mode or interactive mode. Processing differences make the QUEUE language more suitable for batch use and the QUEST language more suitable for interactive use. This does not imply, however, that the QUEST language should not be used in batch or that QUEUE should not be used interactively. As a matter of fact, the QUEST language can be used exclusively in batch mode. The decision as to which language to use depends more on the application itself and, of course, on what processing mode is available to you.

Both QUEST and QUEUE can be used in a single-user environment or in a Multi-User environment. In a single-user environment, each user has an individual copy of SYSTEM 2000 code. In a Multi-User environment, multiple users share a single copy of SYSTEM 2000 code, regardless of whether they are submitting jobs in batch mode or interactive mode. The SYSTEM 2000 Multi-User Facility ensures data and structural integrity, allowing multiple users to retrieve and update a data base concurrently. See *SYSTEM 2000 QUEST Language, Release 11.6 under IBM and CMS* for usage considerations.

The QUEUE processor queries and updates a SYSTEM 2000 data base. Its commands retrieve data from a data base, update data in a data base, and control the execution of retrieval and update commands.

QUEUE language commands are issued in batches during QUEUE/TERMINATE sessions, therefore, making processing of a large amount of data more efficient. You initiate a session by issuing the QUEST language command QUEUE, and you end a session by issuing the QUEUE language command TERMINATE. You can also cancel a session by issuing the QUEUE language command CANCEL QUEUE.

QUEUE language commands are composed of one or more syntax units that can be combined in various ways. The construction of each command is referred to as its syntax. This manual describes the syntax of each QUEUE command, as well as the information needed to formulate meaningful commands.

QUEUE language commands can be issued only between the QUEUE command and the TERMINATE command. Any other commands that can be issued in a QUEUE/TERMINATE session are system-wide commands, although you cannot issue system-wide commands that invoke other processors, such as the QUEST command, the DEFINE command, and the CONTROL command.

The command syntax is flexible, providing you with many options. Several options have default settings, which are automatically set at the beginning of each session. For example, output from retrieval commands is single spaced, unless you specify otherwise. The software also allows you to specify a variety of formats for dates. See the DATE FORMAT IS command in *SYSTEM 2000 QUEST Language*.

The following considerations apply to all SYSTEM 2000 commands:

- They can begin or end in any column.
- Spacing is free-form; that is, commands can have any number of blanks between words.
- Commands can continue from one line to the next.
- More than one command can be entered on a line.
- Each command must end with a colon.
- Component names and component numbers (the letter C followed by the component number) are interchangeable, except in value streams and the Data File, where only the component number (without the preceding C) is used.

The following considerations apply to all QUEUE language commands:

- Each value specified in a command must have the system separator (the default is the asterisk) immediately following it.
- They must all be issued after the QUEUE command.
- They are not executed until the TERMINATE command is issued.
- CALC items cannot be updated if the UNIQUE option is in effect. (See *SYSTEM 2000 QUEST Language* for information on CALC and UNIQUE.)
- Collect File items are not allowed in the QUEUE language.

GENERAL FORMAT FOR COMMANDS

This section presents the general format for QUEUE language retrieval and update commands and tells you where in this manual each form is described in detail.

There are basically two types of retrieval and update commands: tree and non-tree commands. All tree and non-tree commands consist of an *action-clause* and a *where-clause*. The only exception is one form of the APPEND TREE command, which cannot include a where-clause. See **Adding New Data Trees: APPEND TREE** on page 6-2.

The basic format for both tree and non-tree commands is:

```
action-clause where-clause;
```

In an action-clause, you specify the action the command to be performed. In a retrieval command, the action-clause indicates the data to be retrieved. In an update command, the action-clause identifies the item or record to be updated, and includes any new values to be entered. The format for the action-clause varies from command to command. The general format is described in Chapter 3, "Specifying Criteria in Action-Clauses: The If-Clause."

The where-clause is the expression or combination of expressions representing the qualification criteria for a command. It is included in all tree and non-tree commands, except for the command APPEND TREE C0. The syntax and format of the where-clause are described in Chapter 2, "Specifying Qualification Criteria: The Where-Clause."

OVERVIEW OF ACTION-CLAUSES

Action-Clause in Tree Commands

The action-clause of a tree command starts with one of these verbs:

- APPEND TREE, see **Adding New Data Trees: APPEND TREE** on page 6-2.
- PRINT TREE, see **Retrievals from Data Trees: PRINT TREE** on page 4-5.
- REMOVE TREE, see **Removing Data Trees: REMOVE TREE** on page 6-6.

Action-Clauses in Non-Tree Commands

The action-clause of a non-tree command can have any one of these three formats:

1. *non-tree actions*
2. *if-clause* THEN *non-tree actions*
3. *if-clause* THEN *non-tree actions* ELSE *non-tree actions.*

A non-tree action starts with one of these verbs:

- ADD, see **Replacing a Null with a Value: ADD** on page 5-3.
- ASSIGN, see **Replacing a Value or a Null with a Value: ASSIGN** on page 5-10.
- CHANGE, see **Replacing a Value with a New Value: CHANGE** on page 5-7.
- PRINT, see **Retrievals from Data Records: PRINT** on page 4-1.
- REMOVE, see **Replacing a Value with a Null: REMOVE** on page 5-13.

The syntax and format of the if-clause are described in Chapter 3, "Specifying Criteria in Action-Clauses: The If-Clause."

PROCESSING TERMINOLOGY

The terms defined in this section will help you to understand QUEUE command processing.

Target Record

The target record is identified by the component listed to the right of the action verb in retrieval or update commands. A component can be identified by component name or component number. The target record is implicitly identified when you specify an item within a record. There can be only one target record for each command.

The following examples show the target record for each command. The data is drawn from the EMPLOYEE data base, shown in the foldout at the end of this manual.

```
APPEND TREE C0 EQ 1*1050*2*AMEER*3*DAVID*. . .*END*:
```

The target record is C0.

```
APPEND TREE C200 EQ 201*FORTRAN*202*GOOD*203*1*END*WHERE C1 EQ1120*:
```

The target record is C200.

```
REMOVE TREE C420 WHERE C1 EQ 1043*:
```

The target record is C420.

```
PRINT TREE C100 WHERE C1 EQ 1043*:
```

The target record is C100.

```
PRINT C2, ADD C16 EQ 78703* WHERE C1 EQ 1043*:
```

The target record is C0, because C2 and C16 are items in C0.

```
IF C3 EQ MOLLY I.* THEN PRINT C1, CHANGE C16 EQ 78752* ELSE PRINT C0
   WHERE C2 EQ GIBSON*:
```

The target record is C0, because all the components in the action-clause are in that record.

```
IF C105 EQ 10/01/1978* THEN PRINT C1 WHERE C2 EQ GIBSON*:
```

This command is invalid. The action-clause specifies items in different schema records, that is, C105 is not in the same record as C1.

Qualified Data Records

Qualified data records result from the processing of the conditions in the where-clause. The qualified data records for a where-clause are all occurrences of the same schema record. That schema record is called the *focal record* for the where-clause. For a tree or non-tree command to be valid, the target record must be related to the focal record.

Selected Data Records

The selected data records are occurrences of the target record and are related to the qualified data records. In the case of APPEND TREE commands, the selected data records are those occurrences of the parent of the target record that are related to the qualified data records.

The following example shows uses of these terms in analyzing a command. The command refers to Data Entry Two in the foldout at the end of this manual.

```
PRINT C101, PRINT C102 WHERE C2 EQ REID*:
```

The target record is C100. The focal record is C0 because the condition specifies C2, which is an item in C0. The qualified data record is data record 12. The selected data records are those C100 records that are related to data record 12: data records 13 and 16. The values for C101 and C102 are retrieved from those two records when the command is executed.

INITIATING A SESSION: QUEUE

The QUEST language QUEUE command makes the QUEUE processor available to you.

The QUEUE command initiates a QUEUE language session. Once that command is issued, the software can process only commands recognizable by the QUEUE processor. As each command is issued, the software examines it for syntax errors, although it does not execute any commands until the TERMINATE command is issued. The TERMINATE command ends a QUEUE language session.

To protect against damage, the software clears the data base pages when the QUEUE command is issued.

Format

```
QUEUE:
```

Abbreviation: QUEUE = Q

PROCESSING RETRIEVAL AND UPDATE COMMANDS

The main characteristic of the QUEUE processor is that the entire command stream, which begins with the command QUEUE and ends with the TERMINATE command, is read before any retrieval or update occurs. (Some commands are executed upon being issued. See **Processing System-Wide Commands** on page 1-12 and Chapter 9, "Controlling Command Processing " for more information.)

The software performs three distinct operations while processing the commands:

1. scanning for syntax errors
2. executing the commands
3. producing output.

These operations are described below.

Scanning for Syntax Errors

Execution of commands does not occur until each command within the QUEUE/TERMINATE session has been scanned for syntax errors. If the command CANCEL QUEUE is found, scanning stops and you are returned to the QUEST processor. (See Chapter 9, "Controlling Command Processing " and **Cancelling a Session: CANCEL QUEUE** on page 9-5 for more information.)

As each command is scanned, it is assigned a sequence number. A sequence number is also assigned to a command that is repeated because of its inclusion in a REPEAT command. The QUEUE processor displays only the sequence numbers assigned to PRINT TREE commands and to non-tree commands that include a PRINT action. The message identifying the sequence number is displayed with the echo of the command in the Message File. The sequence number is also displayed with the printed output after TERMINATE has been issued, to correlate the output with the original command.

If a command has errors, an error message is written to the Message File. The command is not executed at TERMINATE time; however, the QUEUE processor continues scanning the command stream. You have the option of specifying that scanning be stopped if a specified number of errors occur. See Chapter 9, "Controlling Command Processing " for more information on the STOP IF commands.

Executing the Commands

Once the command stream has been completely examined for syntax errors (that is, after the TERMINATE command is read), the QUEUE processor begins to execute the commands. The execution does not occur if you have issued the DISABLE EXECUTION command or the STOP AFTER SCAN command. See Chapter 9, "Controlling Command Processing " for more information on both commands.

The first step in command execution is to process the where-clauses. (The software keeps track of which where-clause goes with which command by the sequence number of the command.) Where-clause processing is done by using the index, CALC mode (if any), and the information found in the Hierarchical Table. One implication of this process is that only CALC items with the EQ operator and key items can be used in where-clauses of QUEUE language commands. At each step, the information to be used in subsequent steps is kept in scratch files. The result of where-clause processing is a scratch file that contains the Hierarchical Table address of each selected data record. As before, each address is tagged with the corresponding command sequence number.

After the selected data records are found, operations on the data base are carried out in the order shown below.

Illus. 1.1 **Order of Execution of QUEUE Commands**

1. All PRINT TREE operations are executed.
2. All REMOVE TREE operations are executed.
3. All APPEND TREE operations are executed in the order in which they occur in the command stream.
4. All non-tree operations are executed in the order in which they occur in the command stream and in the order in which they occur within a single command, subject to the testing of any associated if-clauses.

Implications of Command Execution Order The order of command execution has the following implications:

1. PRINT TREE commands retrieve values that were in the data base at the time the QUEUE command was issued. That is, modifications caused by update command(s) issued during the QUEUE/TERMINATE session in which the PRINT TREE commands are issued would not be reflected in the output of those commands.

   For example, if a QUEUE/TERMINATE session consists of the commands given below, the output will show values in the data trees topped by C100 records for employee number 2105. These data trees are subsequently removed by the REMOVE TREE command. Notice that the order in which the commands are issued does not affect the final result; PRINT TREE commands are always executed first.

```
QUEUE:
REMOVE TREE C100 WHERE C1 EQ 2105*:
PRINT TREE C100 WHERE C1 EQ 2105*:
TERMINATE:
```

2. Except for PRINT TREE operations, all operations that would otherwise have referred to a data record are ignored if the data record is in a tree that has been removed as a result of a REMOVE TREE command in that QUEUE/TERMINATE session. For example, if the QUEUE/TERMINATE session consists of the commands given below, the PRINT action would not retrieve any values because the REMOVE TREE command would have already removed the C100 records containing the values that were requested by the retrieval action. Notice, again, that the order in which the commands are issued does not affect the final result: execution of REMOVE TREE commands precedes the execution of all APPEND TREE commands and non-tree commands in the session.

```
QUEUE:
PRINT C101 WHERE C1 EQ 2105*:
REMOVE TREE C100 WHERE C1 EQ 2105*:
TERMINATE:
```

3. In executing QUEUE language commands, the where-clauses for all commands are processed prior to their corresponding action-clauses. Therefore, data records in data trees added to the data base via APPEND TREE commands cannot be accessed by other commands in the same QUEUE/TERMINATE session.

   For example, assume that a session consists of the following commands:

```
QUEUE:
APPEND TREE C110 EQ 111*3.50*112*HOURLY*END*
   WHERE C1 EQ 1265*:
PRINT C111 WHERE C1 EXISTS:
APPEND TREE C120 EQ 121*04/30/1979*END*
   WHERE ALL OF (C1 EQ 1265*, C111 EQ 3.50*):
TERMINATE:
```

   The first APPEND TREE command adds a C110 record to Data Entry Three. (See the foldout at the end of this manual.) The second command retrieves all C111 values; however, it cannot retrieve the value for C111 (3.50) entered with the previous command because the where-clause conditions C1 EQ 1265* and C1 EXISTS are processed prior to the execution of the action-clauses. Therefore, the data record added to the data base with the first APPEND TREE command cannot become a selected record for the second command. Likewise, the second APPEND TREE command cannot add the specified C120 record because no C110 record can be selected as parent for this new C120 record.

4. After all PRINT TREE and REMOVE TREE commands are executed, APPEND TREE commands are executed in the order in which they occur in the command stream. Therefore, that order determines the placement of new data records under a parent record. This situation is explained in more detail in **Adding New Data Trees: APPEND TREE** on page 6-2.

5. After all other commands are executed, non-tree commands are executed in the order in which they occur in the command stream. That order determines when the actions specified in each command occur. For example, suppose a session consists of the following commands:

```
QUEUE:
PRINT C16, PRINT C2 WHERE C1 EQ 1043*:
CHANGE C16 EQ 78753* WHERE C1 EQ 1043*:
PRINT C16, PRINT C2 WHERE C1 EQ 1043*:
TERMINATE:
```

The first non-tree command retrieves the values for C16 and C2 in that order, the second command changes the value for C16 in the same record, and the third command retrieves the new value for C16 as well as the value for C2. Because of the order in which the commands are executed, the three commands shown above are equivalent to this command:

```
PRINT C16, PRINT C2, CHANGE C16 EQ 78753*,PRINT C16, PRINT C2
   WHERE C1 EQ 1043*:
```

6. After selected data records for a non-tree command are arranged according to their Hierarchical Table address, they are processed in that sequence by the action-clause of the command. For example, consider the following command, which refers to Data Entry Two in the foldout:

```
IF C114 LE 200.00* THEN PRINT C113 ELSE PRINT C114
   WHERE C1 EQ 1120*:
```

The output from this command is

```
114*       $215.40   → from record 14
113* 03/01/1977      → from record 15
113* 02/16/1976      → from record 17
114*       $239.65   → from record 38
```

The if-clause first checks the value for C114 in record 14. Since that value is greater than 200, the if-clause condition is false. Thus, the alternate action takes place, that is, PRINT C114. Next, the if-clause checks the value for C114 in record 15. That value is less than 200; therefore, the if-clause condition is true and the action PRINT C113 occurs. The same process applies to record 17. Finally, the value for C114 is checked in record 38. Because it is greater than 200, the alternate action occurs.

Producing Output

Data retrieved as a result of PRINT TREE or PRINT operations is sorted and grouped by command sequence number before being sent to the Report File. If the Message File and the Report File are the same physical file, a message indicating the command sequence number precedes the output from the corresponding command. If the TERMINATE command is issued with the UNLOAD options, that is, TERMINATE/UNLOAD, no command sequence number is shown. (See **Ending a Session: TERMINATE** on page 9-3 for more information on TERMINATE/UNLOAD.)

PROCESSING SYSTEM-WIDE COMMANDS

When a system-wide command is issued in a QUEUE/TERMINATE session, it is processed immediately. However, the output from retrievals is always written to the last Report File specified. The commands CONTROL, QUEST, and DEFINE cannot be issued within a QUEUE/TERMINATE session.

COMPARING QUEST AND QUEUE PROCESSING

This section explains the main difference between QUEST processing and QUEUE processing.

SYSTEM 2000 software processes a set of QUEST language commands by executing each command individually and completely before proceeding to the next command. On the other hand, the software processes all the commands in a QUEUE/TERMINATE session at one time in a batch.

In general, processing a QUEST command involves several steps: scanning for syntax errors, accessing the index, CALC records, and Hierarchical Table to process the where-clause (if the command includes one), and using the Hierarchical Table and Data Table to process the action-clause, thereby updating or retrieving data as specified. In QUEUE, all commands go through the first step, then all where-clauses are processed, and finally the software makes one pass through the data base, updating and/or retrieving data as specified. CALC records can be retrieved but the CALC item cannot be updated if the UNIQUE option is set. Collect File items cannot be used in the QUEUE language.

There are two reasons why QUEUE processing requires more scratch files than QUEST processing. In QUEUE processing, the results from one step must be collected in scratch files before the next step begins, and in each step all commands are processed. Also, the results of where-clause processing are always written to a scratch file. In QUEST, scratch file space is required for the results of only one where-clause. But, in QUEUE, scratch file space is required for the results of all where-clauses in the session.

For a more detailed comparison of the two languages see Chapter 11, "Comparing the QUEST and QUEUE Languages."

Chapter 2

Specifying Qualification Criteria: The Where-Clause

INTRODUCTION

This chapter provides the syntax, format, and considerations that apply to the where-clause used in all QUEUE language retrieval and update commands, except for the APPEND TREE C0 command, which cannot specify a where-clause. This chapter also describes the various operators used to form where-clauses. You need to understand the concepts introduced in Chapter 1, "Using the QUEUE Language: Concepts" to understand this chapter.

The where-clause specifies qualification criteria to be used for selecting a subset of the data base for retrieval or update.

- The unary operator EXISTS verifies the existence of values for an item.
- The binary operators (EQ, NE, LE, LT, GE, and GT) compare data items with values.
- The HAS operator qualifies data records according to the contents of their descendant records.
- The threshold operators (ANY OF, ALL OF) combine qualification criteria.
- Only key items and CALC items with the EQ operator can be specified in a where-clause.

THE WHERE-CLAUSE

A where-clause can consist of up to seven *expressions* following the keyword WHERE. Each expression, in turn, is made up of no more than seven *conditions* on data items. Each condition states a criterion that a data item must meet. A where-clause is constructed by specifying one condition, one expression, or by using the threshold operators to combine up to seven expressions.

A condition is specified with one of the following operators: EXISTS, EQ, NE, LE, LT, GE, and GT (or their equivalent symbols). As the software processes a condition, it forms a list of the data records (actually, of their Hierarchical Table addresses) that contain data items satisfying the condition. These records are called the *qualified data records* for the condition. If the condition includes the HAS operator, the qualified records are those occurrences of the record specified before the HAS that are also ancestors of the data records containing the desired values for the items. Each condition refers to a record. This final list consists of records that are occurrences of one schema record (in some cases, ancestors of this schema record, too). This schema record is called the focal record for the where-clause. A focal record can be found only when all of the conditions in the where-clause refer to related records.

If the condition includes the HAS operator, the condition refers to the record specified before the HAS operator. If the condition does not include the HAS operator, the condition refers to the record that contains the item specified in the condition.

An expression is constructed by specifying either one condition or by using the threshold operators ANY OF and ALL OF to combine up to seven conditions. These operators act on the results (lists) produced by the processing of the conditions and derive new lists of qualified records.

When the entire where-clause has been processed, the records in the family of each qualified record become candidates for selection; that is, the records can be selected for any action specified in the action-clause. Thus, each qualified data record serves as a focus for action: the qualified record, all of its descendants, and all of its ancestors can be selected for retrieval or update as specified in the action-clause. Therefore, the target record must be related to the focal record.

Format

```
WHERE |expression
      |ALL  OF expression
      |ANY
```

```
expression   |condition
             |ALL              OF conditions
             |ANY number

condition    |record HAS keyitem  |EXISTS
                                  |binaryoperator value *
```

number is an integer equal to or greater than one but less than or equal to the number of expressions or to the number of conditions within the parentheses that follow the ANY OF operator. If *number* is not specified, the default is 1.

record is the name or component number of a schema record that is an ancestor of the record containing the key item.

keyitem is the name or component number of a key item.

Abbreviations:

EXISTS = EXIST, EXISTING
HAS = HAVE, HAVING
WHERE = WH

binaryoperator EQ, NE, LE, LT, GE, GT, or the following symbols:

Operator	Symbols
EQ	=
GE	>= or => or < or !<
GT	>
LE	<= or =< or > or !>
LT	<
NE	= or !=

value is a literal value, or one of the two system strings *TODAY* or *DATA*.

Considerations

- You must include a where-clause in every retrieval or update command, unless the command is APPEND TREE C0. In that case, a where-clause is not allowed.
- All items specified in a where-clause must be key items.
- Each value specified in a where-clause must have a system separator immediately following it. The default system separator is the asterisk.
- The HAS operator must be followed by an item name or component number.
- All conditions in a where-clause must refer to related records.
- No more than seven expressions can be specified within the parentheses following the keyword OF. No more than seven conditions can be specified within the parentheses following the keyword OF. Therefore, a where-clause cannot include more than 49 conditions.
- The threshold operators ANY OF and ALL OF can be nested only once. For example, a where-clause of the following form is invalid.

```
. . . WHERE ANY OF (condition1, ALL OF condition2, condition3,
ANY OF (condition4, condition5)), condition6):
```

- The maximum number of characters allowed for any value specified in a where-clause is 250.

- If Security by Entry is in effect for a data base, a secondary password holder must establish the "area" of access before entering the QUEUE processor because it is impossible to establish or change the "area" within a QUEUE/TERMINATE session. Refer to *SYSTEM 2000 CONTROL Language and System-Wide Commands, Release 11.5 under IBM OS and CMS* for more information about Security by Entry.

- Only items and records for which you have W-authority can appear in a where-clause.

- If a where-clause condition contains a CALC item with the EQ operator, CALC mode processing is used to locate the qualified record(s).

- Collect File items are not allowed in the QUEUE language where-clause. (Collect File items relate to the QUEST language.)

- For non-tree commands, you can place conditions in the if-clause instead of the where-clause. For details, see the discussion in Chapter 3, "Specifying Criteria in Action-Clauses: The If-Clause" under uses of the if-clause. Also, Illus. 3.1 on page 3-5 compares if-clause syntax with where-clause syntax.

- For the use of where-clauses in retrieval or update commands, refer to the discussion on the appropriate command in Chapter 4, "Retrieving Data," Chapter 5, "Updating Data Records" and Chapter 6, "Updating Data Trees."

- Without symbols, the software reads a blank on either side of an operator to separate it from other syntax units. However, with symbols, blanks are not required for separation. Any blanks encountered following a symbol are used as part of the value. For example, these two where-clauses are equivalent:

```
WHERE C1 EQ ABC AND C2 GT DEF:
WHERE C1 =ABC&C2>DEF:
```

- The EQ operator and = symbol are not interchangeable in an update action-clause. You must use EQ when specifying values in ADD, CHANGE, ASSIGN, and INSERT commands. However, if the value is a computation, you must use the = symbol.

- If component names contain any of the available symbols, use the component numbers. Also, if you change the system separator to be any of the symbols, syntax errors could occur if you specify that symbol in the where-clause.

Optimization

The following information presents techniques that you can use for optimizing QUEUE language where-clauses.

One characteristic of QUEUE where-clause processing is that a list of qualified data records is generated for each condition. When possible, conditions that generate long lists should be placed in the if-clause. (See the discussion on uses of the if-clause in Chapter 3, "Specifying Criteria in Action-Clauses: The If-Clause.") Also, keep in mind that conditions using the EXISTS operator tend to generate longer lists than other conditions. Often, if several consecutive commands include the same condition with the EXISTS operator, you should consider using QUEST language commands with the SAME operator. For example, consider the following QUEUE language commands.

```
PR C1 WH C2 EXISTS:
PR C101 WH ALL OF (C2 EXISTS, C102 EXISTS):
PR C122 WH ALL OF (C2 EXISTS, C102 EXISTS, C121 EXISTS):
```

The same results can be obtained with the following three QUEST language commands that use the SAME operator. This avoids the creation of three separate lists for C2 EXISTS and of two separate lists for C102 EXISTS.

```
PR C1 WH C2 EXISTS:
PR C101 WH SAME AND C102 EXISTS:
PR C122 WH SAME AND C121 EXISTS:
```

Besides reducing the size or the number of lists created during where-clause processing, it is also important to consider reducing the amount of processing involved. Two considerations apply to reducing processing time.

- Where possible, use the HAS operator to avoid automatic downward normalization. (See the HAS operator in *SYSTEM 2000 QUEST Language, Release 11.6 under IBM and CMS*.) For example, it is more efficient to process the command

  ```
  PR C1 WH ANY OF (C2 EQ BROWN*, C0 HAS C201 EQ COBOL*):
  ```

  than to process this command:

  ```
  PR C1 WH ANY OF (C2 EQ BROWN*, C201 EQ COBOL):
  ```

  In the first command, the selected C0 data records are the same as the qualified data records for the where-clause. Normalization is required only to process the second condition. In the second command, normalization is required to obtain the selected C0 records from the C200 qualified data records and to determine the C200 records that are descendants of the C0 records satisfying C2 EQ BROWN.

- When multiple conditions or expressions are operated on by ANY OF or ALL OF, processing time can be saved if conditions likely to fail are placed at the leftmost position for ALL OF and if conditions likely to succeed are placed at the left for ANY OF. This is because multiple conditions or expressions within ANY OF or ALL OF are processed from left to right and testing for qualified records stops as soon as the result is known. This consideration also applies to ANY OF and ALL OF expressions in the if-clause. For example, if you suspect that no employee was hired on March 1, 1979, the following where-clause expression

  ```
  ALL OF (C4 EQ 03/01/79*, C8 EQ CAUCASIAN*, C102 EQ MARKETING*)
  ```

  would be more efficient to process than the equivalent expression

  ```
  ALL OF (C102 EQ MARKETING*, C8 EQ CAUCASIAN*, C4 EQ 03/01/79*)
  ```

Examples

The examples in this section show the syntax rules for constructing where-clauses; therefore, the commands use simple action-clauses. All commands refer to the EMPLOYEE data base shown in the foldout at the end of this manual.

Example 1

In this command, C2 is a key item. C0 is both the focal record and the target record; therefore, the command is valid.

```
PRINT C1 WHERE C2 EXISTS:
```

Example 2

In this command, C2 and C7 are key items. C0 is both the focal record and the target record; therefore, the command is valid. The where-clause consists of one expression made up of two conditions.

```
PRINT C1 WH ALL OF (C2 EXISTS, C7 EQ MALE*):
```

Example 3

In this command, C2, C7, and C102 are key items. The where-clause consists of one expression made up of three conditions. The first two conditions refer to record C0. The last condition refers to record C100. C0 and C100 are related; therefore, the where-clause is valid. The focal record is C100. (See **Where-Clause Processing** on page 2-19 for an explanation of how the focal record is determined.) The target record is C0. Since the target record and the focal record are related, the command is valid.

```
PR C1 WH ANY OF (C2 EXISTS, C7 EQ MALE*, C102 EQ MARKETING*):
```

Example 4

In this command, C7 and C201 are key items. The where-clause consists of one expression made up of two conditions. Both conditions refer to record C0 (the first one because the item is in C0, the second one because C0 is the record specified before HAS). The where-clause is valid, and its focal record is C0. The target record is C100. Because C0 and C100 are related, the command is valid.

```
PR C101 WH ANY OF (C7 EQ MALE*, C0 HAS C201 EQ COBOL*):
```

Example 5

In this command, C7, C102, C111, C112, and C113 are key items. The where-clause consists of three expressions: the first expression consists of only one condition that refers to record C0, the second expression is made up of two conditions that refer to record C100, and the third expression is made up of three conditions that refer to record C100. Because C0, C100, and C110 are related, the where-clause is valid. The focal record is C110. The target record is C120. Since C110 and C120 are related, the command is valid.

```
PR C120 WH ALL OF (C7 EQ MALE*,
   ANY OF (C102 EQ MARKETING*, C102 EQ INFORMATION SYSTEMS*),
   ANY 2 OF (C111 EQ 1000.00*,C112 EQ MONTHLY*,C113 EQ 01/01/75*)):
```

Example 6

In this command, C102 and C412 are key items. The where-clause consists of two expressions: the first expression is made up of one condition that refers to record C0 (the record before HAS),and the second expression is made up of three conditions that refer to record C400 (the record before HAS). Because C0 and C400 are related, the where-clause is valid. The focal record is C400 and the target record is C0; therefore, the command is valid.

```
PR C1 WH ALL OF (C0 HAS C102 EQ MARKETING*,
   ANY 2 OF (C400 HAS C412 EQ MA*,C400 HAS C412 EQ MS*,
   C400 HAS C412 EQ MBA*)):
```

Example 7

In this command, C102, C412, and C413 are key items. The where-clause consists of four expressions. The first expression consists of one condition that refers to record C0, and the second and third expressions consist of two conditions each. All four conditions refer to record C410. The fourth expression consists of one condition that refers to record C410. Because C0 and C410 are related, the where-clause is valid. The focal record is C410 and the target record is C0; therefore, the command is valid.

```
PR C1 WH ANY 2 OF (C0 HAS C102 EQ MARKETING*,
   ALL OF (C412 EQ BA*, C413 GE 05/31/77*),
   ALL OF (C412 EQ BS*, C413 GE 05/31/77*),
   C412 EQ MBA*):
```

SPECIFYING CONDITIONS

As mentioned, a where-clause consists of one or more conditions. Conditions specify criteria that must be met by an item in order for it to qualify for further processing. A condition always includes an item, a keyword (called an operator), and, depending on the operator, a value.

There are seven operators that can form conditions. Six are called *binary* operators, because they have two operands: a key item and a value. The seventh operator, EXISTS, is called a unary operator because it has only a single operand: a key item. Notice that only key items can be specified. This is because QUEUE processing of where-clauses uses the index created for key items. (CALC keys can also be used in QUEUE processing of where-clauses. See *SYSTEM 2000 QUEST Language, Release 11.6 under IBM and CMS* and *SYSTEM 2000 DEFINE Language, Release 11.5 under IBM OS and CMS* for information on CALC keys.) In contrast, because of the difference in processing, items specified in the where-clauses of QUEST language commands can be either key or non-key items. Collect File items cannot be used in the QUEUE language.

The unary operator EXISTS specifies that only those records that have a value for the item are to be considered for processing. For example, the following command retrieves the last names of only those employees whose JOB SKILLS records contain a value for SKILL TYPE.

```
PRINT LAST NAME WHERE SKILL TYPE EXISTS:
```

The binary operators (EQ - equals, NE - not equals, LE - less than or equal, LT - less than, GE - greater than or equal, and GT - greater than) are used to compare a data item with a value. For example, a command such as the one below retrieves the last names of only those employees whose JOB SKILLS records contain the value COBOL for SKILL TYPE. Notice that the system separator (that is, the asterisk) follows the value specification when a binary operator is used.

```
PRINT LAST NAME WHERE SKILL TYPE EQ COBOL*:
```

As the software processes a condition, it forms a list of Hierarchical Table addresses of the records that contain values satisfying the condition, that is, qualified data records. For example, the qualified data records that result from the processing of the condition SKILL TYPE EQ COBOL* are those C200 records that contain the value COBOL for SKILL TYPE.

The previous examples might give the impression that only one condition can be included in a where-clause. Actually, while in some cases one condition might suffice to express the desired qualification criteria, complex criteria can require a where-clause consisting of several conditions. In some cases, the HAS operator must also be included in a condition. (See **Specifying HAS in a Condition** on page 2-13.)

Specifying the Existence of Values Using EXISTS

The unary operator EXISTS tests for the existence of values.

EXISTS is called a unary operator because it has only a single operand: a key item. EXISTS qualifies all data records that have a value for the specified item.

Only key items can be used to test the existence of values. Records cannot be used.

Format

keyitem **EXISTS**

keyitem is the name or component number of a key item.

Example The following command requests the last names of employees who have values for SECURITY CLEARANCE:

```
PRINT LAST NAME WHERE SECURITY CLEARANCE EXISTS:
```

Abbreviations: EXISTS = EXIST, EXISTING

Comparing a Data Item with a Value: EQ, NE, GE, GT, LE, or LT

The binary operators EQ, NE, GE, GT, LE, and LT (or their equivalent symbols) compare data items in the data base with a given value.

Binary operators have two operands: an item and a specified value.

EQ qualifies the data records that contain the specified value for the item. (CALC mode processing is used for conditions that contain a CALC item with the EQ operator.) For example, the following where-clause qualifies those C0 data records where the value for EMPLOYEE NUMBER is equal to 1224.

```
. . . WHERE EMPLOYEE NUMBER EQ 1224*:
```

NE qualifies those data records that contain a value for the item that is not equal to the specified value. For example, this where-clause qualifies those C0 records where EMPLOYEE STATUS is not equal to FULL TIME. It does not qualify those records where EMPLOYEE STATUS is null.

```
. . . WHERE EMPLOYEE STATUS NE FULL TIME*:
```

GE qualifies those records that contain values for the item that are greater than or equal to the specified value. For example, the following where-clause qualifies those C0 records where the value for HIRE DATE is greater than or equal to June 1, 1977. Thus, an employee hired on or after that date would qualify.

```
. . . WHERE HIRE DATE GE 06/01/1977*:
```

GT qualifies those records that contain values for the item that are greater than the specified value. For example, the next where-clause qualifies those records where the value for HIRE DATE is greater than June 1, 1977. Thus, an employee who was hired on that date would not qualify, but all employees hired after that date would.

```
. . . WHERE HIRE DATE GT 06/01/1977*:
```

LE qualifies those records that contain values for the item that are less than or equal to the specified value. For example, the next where-clause qualifies those records where the value for HIRE DATE is less than or equal to June 1, 1977. Thus, an employee hired on or before that date would qualify.

```
. . . WHERE HIRE DATE LE 06/01/1977*:
```

LT qualifies those records that contain values for the specified item that are less than the specified value. For example, the next where-clause qualifies those records where the value for HIRE DATE is less than June 1, 1977. Thus, an employee who was hired on or after that date would not qualify, but all employees hired before the date would.

```
. . . WHERE HIRE DATE LT 06/01/1977*:
```

Format

```
keyitem |EQ value systemseparator
        |NE
        |GE
        |GT
        |LE
        |LT
```

keyitem is the name or component number of a key item.

value is a literal value or one of the two system strings *DATA* or *TODAY*. (See *SYSTEM 2000 QUEST Language* for information on system strings.)

systemseparator is a single special character. The default is the asterisk.

Considerations

- An asterisk (the default system separator) always follows the value specified. If *DATA* is used, no asterisk is specified after it. For example, the asterisk is required in the following condition:

  ```
  C102 EQ MARKETING*
  ```

  However, in this condition, the use of another asterisk is invalid:

  ```
  C102 EQ *DATA*
  ```

- When a value is specified for a TEXT item, the blank space after the binary operator acts as a delimiter for the operator and is not considered part of the value. However, all blanks after the first and before the asterisk are considered part of the value. For example, this next condition is satisfied by those C130 records that have the value ␣ABC␣␣ for item C132. This example assumes that C132 is a key item.

  ```
  C132 EQ␣␣ABC␣␣*
  ```

- CALC mode processing locates the qualified record(s) if the condition contains a CALC item with the EQ operator.

SPECIFYING HAS IN A CONDITION

The HAS operator allows you to explicitly identify a record as the focal record for a condition.

It causes the software to replace the list of data records meeting the condition for an item with a new list. This new list consists of those data records' ancestors that are occurrences of the schema record specified to the left of HAS.

For example, in the EMPLOYEE data base, the HAS operator in the following condition causes the C200 records containing the value of COBOL for item C201 to be replaced by their parent C0 records.

```
C0 HAS C201 EQ COBOL*
```

The HAS operator is used for three reasons:

1. to construct where-clauses so that all conditions refer to related records.
2. to combine exclusive conditions when the ALL OF operator or the ANY *number* OF operator is used with *number* larger than 1.
3. to raise the level of the focal record when a condition refers to a record that is disjoint from the target record.

Examples at the end of this section illustrate these uses.

Format

```
condition [record HAS] keyitem |EXISTS
                               |binaryoperator value *
```

record	is the record name or component number of a schema record that is an ancestor of the record containing the key item.
keyitem	is the name or component number of a key item.
binaryoperator	EQ, NE, LE, LT, GE, or GT (or their equivalent symbols).
value	is a literal value, or one of the two system strings *TODAY* or *DATA*. (See *SYSTEM 2000 QUEST Language* for information on system strings.)

Abbreviations:

EXISTS = EXIST, EXISTING
HAS = HAVE, HAVING

Considerations

- A schema record name or component number is required immediately to the left of the HAS operator.
- A key item name or component number is required immediately to the right of the HAS operator.
- The record to the left of HAS must be an ancestor of the record containing the item specified to the right of HAS.

Examples

Examples 1 through 3 show the uses of the HAS operator listed above. The commands refer to the EMPLOYEE data base.

Example 1

The target record is C100 and the focal record is C0. The command is valid because C0 and C100 are related records. Notice that if the HAS operator were not used, the focal record would be C200 and the command could not be processed because the focal record would be disjoint from the target record.

```
PR C101 WH C0 HAS C201 EQ COBOL*:
```

Example 2

The following three commands show the use of the HAS operator when an expression contains exclusive conditions. The HAS operator is required in the first and third commands because it is impossible for one data record to have all specified values for one item. The HAS operator in the where-clause of the first command causes the software to find the C0 records that have at least one child C200 record with the value of COBOL for C201 and at least one other C200 record with the value of FORTRAN for C201. The HAS operator in the where-clause of the third command causes the software to find C0 records with at least two descendant C200 records and with each descendant containing one of the three specified values. The HAS operator is not needed in the where-clause of the second command because any data record containing either the value COBOL or the value FORTRAN for C201 satisfies the where-clause.

```
PR C1 WH ALL OF (C0 HAS C201 EQ COBOL*, C0 HAS C201 EQ FORTRAN*):

PR C1 WH ANY OF (C201 EQ COBOL*, C201 EQ FORTRAN*):

PR C1 WH ANY 2 OF (C0 HAS C201 EQ COBOL*, C0 HAS C201
   EQ FORTRAN*, C0 HAS C201 EQ PL/I*):
```

Example 3

The commands in this example refer to Data Entry Two in the foldout at the end of this manual. The commands and their respective output are shown first. The table after the output from the last command identifies the target record, the focal record, the qualified data records, and the selected data records for each command.

```
PR C201 WH ALL OF (C0 HAS C105 EQ 03/01/77*, C0 HAS C201 EQ COBOL*):

201* PL/I
201* COBOL
201* FORTRAN

PR C101 WH ALL OF (C0 HAS C105 EQ 03/01/77*, C0 HAS C201 EQ COBOL*:)

101* PROGRAMMER
101* ASSISTANT PROGRAMMER

PR C201 WH ALL OF (C0 HAS C105 EQ 03/01/77*, C201 EQ COBOL*):

201* COBOL

PR C101 WH ALL OF (C105 EQ 03/01/77*, C0 HAS C201 EQ COBOL*):

101* PROGRAMMER
```

	target record	focal[1] record	qualified data records	selected data records
command	C200	C0	12	18, 19, 47
command	C100	C0	12	13, 16
command	C200	C200	19	19
command	C100	C100	13	13

[1] See **Where-Clause Processing** on page 2-19 for details on how to determine the focal record for an expression.

SPECIFYING EXPRESSIONS WITH THRESHOLD OPERATORS: ALL OF, ANY OF

Conditions or expressions in a where-clause are combined by the threshold operators ALL OF and ANY OF. This section describes the syntax for these operators.

The Threshold Operator ALL OF

The ALL OF and the ANY OF operators construct expressions of up to seven conditions or construct where-clauses consisting of up to seven expressions. (The ANY OF operator is explained in the next subsection.) The conditions or expressions to be combined are separated by commas and enclosed in parentheses following the operator.

The following command displays the last names of all black, female employees.

```
PRINT LAST NAME WHERE ALL OF (SEX EQ FEMALE*, ETHNIC ORIGIN EQ BLACK*):
```

The conditions specified in the expression must refer to items in related records; if that is not the case, you must use the HAS operator. You must also use the HAS operator when two or more conditions specify the same item.

The ALL OF operator combines conditions (or expressions) by making a list of those data records that satisfy all conditions (or all expressions) enclosed in the set of parentheses following the operator. This list consists of occurrences of the lowest-level record referred to in the condition (that is, the focal record).

Note that the ALL OF operator is equivalent to the logical operator AND in QUEST language where-clauses.

When exclusive conditions are specified after the ALL OF operator or after an ANY OF operator with threshold larger than one, the HAS operator must be included in the conditions. The format is given in **The Where-Clause** on page 2-2.

The commands in these examples refer to Data Entry Two in the foldout at the end of this manual. Additional examples of ALL OF and ANY OF are also included in **The If-Clause** on page 3-2.

Example 1

This command retrieves the value for C121 in record 37.

```
IF ALL OF (C122 EQ 160.*,C123 EQ 00.*) THEN PR C121 WH C1 EQ 1120*:
```

Example 2

This command retrieves the value of C121 in records 37, 40, and 42.

```
IF ANY 2 OF (C122 EQ 160.*, C123 EQ 00.*, C125 EQ 153.20*)
   THEN PR C121 WH C1 EQ 1120*:
```

Example 3

This command retrieves the value of C121 in record 35.

```
IF ALL OF (C123 GE 10.*, ALL OF (C121 GE 01/01/79*,C121 LE 04/01/89*),
   ANY OF (C122 EQ 184.0*, C122 EQ 160.0*))
   THEN PR C121 WH C1 EQ 1120*:
```

Example 4

Because the if-conditions are exclusive, only the action specified after ELSE takes place. This command retrieves the values for C126 for records 35, 37, 40, 42, and 45.

```
IF ALL OF (C122 EQ 160.0*, C122 EQ 184.0*) THEN PR C121 ELSE PR C126
   WHERE C1 EQ 1120*:
```

The Threshold Operator ANY OF

When any one of several conditions must be true in order to meet the qualification criteria, supply the conditions and tell the software that ANY OF the conditions must be true. You do this by enclosing the conditions in parentheses as follows:

```
PR LAST NAME WH ANY OF (SKILL TYPE EQ COBOL*, SKILL TYPE EQ FORTRAN*):
```

The above command displays the last names of those employees who know COBOL, or who know FORTRAN, or who know both languages. Notice that the ANY OF operator is equivalent to the logical operator OR in QUEST language where-clauses.

For example, the above command produces the same results as the following QUEST language command:

```
PR LAST NAME WH SKILL TYPE EQ COBOL OR SKILL TYPE EQ FORTRAN:
```

When more than one, but not all, of several conditions must be true to meet the qualification criteria, you can supply the conditions and tell the software that ANY *number* OF the conditions must be true. (The number specified within the ANY OF is called the threshold.) You enclose the conditions in parentheses as follows:

```
PR LAST NAME WH ANY 2 OF (C7 EQ FEMALE*, C8 EQ BLACK*, C9 EQ
   FULL TIME*):
```

The above command displays the last names of those employees who meet at least two of the conditions.

The syntax allows up to seven conditions within the parentheses following ANY OF (or ANY *number* OF). The conditions are separated by commas. Notice that if the number specified between ANY OF is equal to the number of conditions within the parentheses, the expression is equivalent to an expression with the same conditions preceded by ALL OF. For example,

```
ANY 3 OF (condition1, condition2, condition3)
```

is equivalent to

```
ALL OF (condition1, condition2, condition3)
```

Also, specifying ANY 1 OF is equivalent to specifying ANY OF.

The *ANY number OF* operator combines conditions (or expressions) by making a list of those data records that satisfy at least that *number* of conditions (or expressions). As explained in **Where-Clause Processing** on page 2-19, this list consists of occurrences of the lowest-level record referred to in the conditions, that is, the focal record. In some cases, the list might include records that are occurrences of ancestors of the focal record. For instance, see Example 4 on the previous page.

As in the case of expressions formed by using ALL OF, the conditions specified in an expression formed by using ANY OF must refer to items in related records. If that is not the case, you must use the HAS operator.

Combining Expressions

A where-clause can consist of a single condition or a single expression. An expression is a set of no more than seven conditions enclosed in parentheses and preceded by one of the threshold operators ALL OF or ANY OF. However, it is possible that a single expression is not sufficient to describe the qualification criteria for the command. In that case, the threshold operators can be used to operate on expressions. In summary, a where-clause can consist of the ALL OF or ANY OF operator followed by up to seven expressions enclosed in parentheses, each expression, in turn, consisting of the ALL OF or ANY OF operator followed by up to seven conditions. In its most complex form, a where-clause can include a maximum of 49 conditions: seven expressions, each with seven conditions.

The following examples show where-clauses that include more than one expression.

1. The following command displays the EMPLOYEE NUMBER (C1) and LAST NAME (C2) of those female employees who know COBOL, FORTRAN, or both.

```
PRINT C1, PRINT C2 WHERE ALL OF (SEX EQ FEMALE*,
   ANY OF (JOB SKILL EQ COBOL*, JOB SKILL EQ FORTRAN*)):
```

2. This command displays the LAST NAME (C2) of all female employees who know at least one of the two programming languages and who are either black or of Hispanic descent.

```
PRINT C2 WHERE ALL OF (SEX EQ FEMALE*,
   ANY OF (JOB SKILL EQ FORTRAN*, JOB SKILL EQ COBOL*),
   ANY OF (C8 EQ BLACK*, C8 EQ HISPANIC*)):
```

3. This command displays the last name of those employees who are female and are either black or Hispanic, or are female and work full time, or are either black or Hispanic and work full time.

```
PRINT C2 WHERE ANY 2 OF (C7 EQ FEMALE*,
   ANY OF (C8 EQ BLACK*, C8 EQ HISPANIC*), C9 EQ FULL TIME*)):
```

As mentioned, all of the conditions in a where-clause must refer to items in related records; if not, the HAS operator must be used. To understand the use of the HAS operator, you need to understand where-clause processing, which is the topic of the next section.

WHERE-CLAUSE PROCESSING

Data records that result from where-clause processing are all occurrences of the same schema record, which is called the focal record for the where-clause. Also, for a command to be valid, the target record must be related to the focal record. This section describes how the focal record for a where-clause is determined and how to use the HAS operator to specify a focal record.

Determining the Focal Record for an Expression

Unless the HAS operator is used, all conditions within an expression must refer to items in related records.

If all items in the conditions within an expression are in the same record, that record becomes the focal record for the expression. For example, the focal record for the following expression is C0, because all items in the three conditions are in that record.

```
ALL OF (C4 GE 01/01/1979*, C7 EQ FEMALE*, C8 EQ CAUCASIAN*)
```

If the items in the conditions within an expression are from different related records, the focal record for the expression is the one at the lowest level. For example, the focal record for this expression is C120, because C7 is in C0, C102 is in C100, C121 is in C120, and C120 is the lowest of the three records. This implies that if an expression consists of conditions referring to items in different related records, the qualified records are occurrences of the lowest of these records. Example 1 on the next page shows how those records are determined.

```
ANY OF (C7 EQ FEMALE*, C102 EQ MARKETING*, C121 EQ 01/31/1979*)
```

Example 1

Consider the following expression and assume it applies only to Data Entry One and Data Entry Two in the foldout at the end of this manual.

```
ALL OF (C8 EQ FULL TIME*, C102 EQ INFORMATION SYSTEM*, C111 GE 1200.00*)
```

The Hierarchical Table addresses (indicated by bold numbers above the records on the data trees) of the C0 records satisfying C8 EQ FULL TIME are 1 and 12. The Hierarchical Table addresses of the C100 records satisfying the condition C102 EQ INFORMATION SYSTEMS are 2, 13, and 16.

The Hierarchical Table addresses of the C110 records that satisfy the condition C111 GE 1200.00 are 3, 14, and 38.

The focal record (that is, the lowest record) is C110. Before finding the qualified records for the expression, the software replaces the Hierarchical Table addresses of the C0 and C100 records by those of their C110 descendants. This process is called *downward normalization*.

Therefore, the list of C0 records (1 and 12) is replaced by the Hierarchical Table addresses of their C110 descendants: 3, 14, 15, 17, and 38. The list of C100 records (2, 13, and 16) is replaced by the Hierarchical Table addresses of their C110 descendants: 3, 14, 15, 38, and 17. Then, the ALL OF operator is applied to these three lists, and those C110 records whose Hierarchical Table addresses appear in all three lists become the qualified data records for this expression:

```
ALL OF  ([3,14,15,17,38] ,[3,14,15,17,38],[3,14,38])
```

Since 3, 14, and 38 appear in all three lists, those are the Hierarchical Table addresses of the C110 qualified data records for the expression.

If the operator had been ANY OF, the Hierarchical Table addresses of the qualified C110 records would have been: 3, 14, 15, 17, and 38.

Determining the Focal Record for a Where-Clause

If a where-clause consists of several expressions and all conditions specify items in the same record, that record becomes the focal record for the where-clause. If the where-clause consists of several expressions and the conditions specify items in different related records, the focal record is the one at the lowest level.

Example 2, on the next page, shows how qualified records for a where-clause are determined.

Example 2

Consider the following where-clause and assume it applies only to Data Entry One and Data Entry Two in the foldout at the end of this manual.

```
. . . WHERE ALL OF (C9 EQ FULL TIME*, C105 GE 01/01/1977*,
    ALL OF (C113 GE 01/01/1978*, C113 LE 12/31/1978*))
```

The Hierarchical Table addresses of the C0 records satisfying the condition C9 EQ FULL TIME are 1 and 12.

The Hierarchical Table addresses of the C100 records satisfying the condition C105 GE 01/01/1977 are 2 and 13.

The Hierarchical Table addresses of the C110 records satisfying the condition C113 GE 01/01/1978 are 3, 14, and 38. The corresponding addresses for the condition C113 LE 12/31/1978 are 3, 14, 15, and 17. Therefore, the C110 records satisfying the following expression are 3 and 14.

```
ALL OF (C113 GE 01/01/1978*, C113 LE 12/31/1978*)
```

The focal record for the where-clause is C110. As in Example 1, the Hierarchical Table addresses of the C0 and C100 records are replaced by the addresses of their descendant C100 records. Therefore, the list of C0 records (1 and 12) is replaced by the addresses of their C110 descendants: 3, 14, 15, 17, and 38. The list of C100 records (2 and 13) is replaced by the addresses of their descendant C110 records: 3, 14, 15, and 38. The ALL OF operator is now applied to all three lists and those C110 records whose addresses appear in all three lists become the qualified data records for the where-clause: 3 and 14. This process is shown below.

Records satisfying conditions *C9 EQ FULL TIME** and *C105 GE 01/01/1979** are:

```
ALL OF ([1,12], [2,13]
```

Records satisfying conditions *C113 GE 01/01/1978** and *C113 LE 12/31/1978** are:

```
ALL OF ([3,14,38], [3,14,15,17]))
```

Applying ALL OF to the last two lists results in:

```
ALL OF ([1,12], [2,13], [3,14])
```

Normalizing down to level 2 results in:

```
ALL OF ([3,14,15,17,38], [3,14,15,38], [3,14]))
```

Applying ALL OF to the three lists results in:

```
[3,14]
```

Notice that the expression *ALL OF (C113 GE 01/01/1978*,C113 LE 12/31/1978*)* is equivalent to the condition *C113 SPANS 01/01/1978*12/31/1978* in a QUEST language where-clause.

The HAS Operator

If the conditions in an expression specify items in disjoint records, the software cannot process the where-clause without using the HAS operator. For example, consider the following command:

```
PRINT C2 WHERE ALL OF (C201 EQ COBOL*, C102 EQ INFORMATION SYSTEMS*):
```

The command requests the last name of those employees in the Information Systems Department who know COBOL. For instance, the employee whose data are shown in Data Entry Two at the end of this manual meets the requirements. However, the where-clause cannot be processed.

Consider Data Entry Two in the foldout at the end of this manual. The Hierarchical Table address of the C200 record satisfying the condition C201 EQ COBOL is 19. The Hierarchical Table addresses of the C100 records satisfying the condition C102 EQ INFORMATION SYSTEMS are 13 and 16. When the ALL OF operator is applied, that is, ALL OF ([19],[13,16]), there is no number that appears in both lists.

However, if those addresses were replaced by the Hierarchical Table address of their parent C0 record, that is, ALL OF ([12],[12]), data record 12 becomes a qualified data record. Therefore, to process expressions consisting of conditions that specify items from disjoint schema records, you need to tell the software to replace the addresses of those disjoint records by the addresses of common ancestor records. (This process is called *upward normalization*.) Do this by specifying the name of component number of the ancestor record, the HAS operator, and the condition.

For example, the command given above becomes

```
PRINT C2 WHERE ALL OF(C0 HAS C201 EQ COBOL*, C0 HAS C102 EQ
   INFORMATION SYSTEM*):
```

The software processes the where-clause as shown below.

```
. . . ALL OF (C0 HAS C201 EQ COBOL*, C0 HAS C102 EQ INFORMATION SYSTEMS*)
```

Records satisfying conditions are:

```
ALL OF (C0 HAS [19], C0 HAS [13,16])
```

Applying HAS for upward normalization results in:

```
ALL OF ([12],[12])
```

Because 12 appears in both lists, it is the address of the qualified record.

When the HAS operator is used in a condition, the record that precedes it is an ancestor of the record containing the item in the condition. Therefore, by choosing appropriate ancestor records, the HAS operator can be used so that all the conditions in the where-clause refer to related records.

The HAS operator must also be used in two other cases: when two or more conditions specify different values for the same item (exclusive conditions), and when the target record and the focal record are disjoint.

Exclusive conditions Consider the following command:

```
PRINT C2 WHERE ALL OF (C201 EQ COBOL*, C201 EQ PL/I*):
```

This command requests the last name of those employees who know both programming languages. For example, the employee in Data Entry Two meets the criterion. However, when the software tries to process the where-clause, no qualified data records are found, and the software does not retrieve any values. (The address of the record satisfying C201 EQ COBOL is 19, the address of the record satisfying C201 EQ PL/I is 18; therefore, when the ALL OF operator is applied, no record is found that qualifies.)

The HAS operator is used in this case to specify the parent record, as shown below:

```
PRINT C2 WHERE ALL OF (C0 HAS C201 EQ COBOL*, C0 HAS C201 EQ PL/I*):
```

The HAS operator must also be used when exclusive conditions are specified and the ANY OF operator is used with a threshold larger than 1. For example, the where-clause in the following command produces no qualified data records:

```
PRINT C2 WHERE ANY 2 OF (C201 EQ COBOL*, C201 EQ FORTRAN*, C201 EQ PL/I*):
```

In contrast, by using the HAS operator, those employees who know any two of those languages can be found.

```
PRINT C2 WHERE ANY 2 OF (C0 HAS C201 EQ COBOL*, C0 HAS C201 EQ PL/I*):
```

Target record disjoint from focal record Consider this command:

```
PRINT C201 WHERE C102 EQ INFORMATION SYSTEMS*:
```

The focal record is C100, and the target record is C200. These records are disjoint, and the software cannot process the command. However, if the condition in the where-clause includes the HAS operator, the command can be processed as shown below:

```
PRINT C201 WHERE C0 HAS C102 EQ INFORMATION SYSTEM*:
```

The focal record is now C0, and because C0 is the parent of the C200 target record, the command is valid and can be processed. It retrieves the C201 values from those C200 records that are descendants of the C0 qualified data records.

In summary, the HAS operator must be used in three cases:

- to construct where-clauses so that all of the conditions refer to related records.
- to combine exclusive conditions when the ALL OF operator or the ANY *number* OF operator is used with *number* larger than 1.
- to raise the level of the focal record when the focal record and the target record are disjoint.

WHERE-CLAUSES: SUMMARY AND EXAMPLES

Except for the APPEND TREE C0 command, a where-clause must be used in all QUEUE language retrieval or update commands.

Where-clauses consist of one or more conditions. Conditions are simple statements about items and are formed by using any of seven operators: EXISTS, EQ, NE, LE, LT, GE, or GT. When necessary, a where-clause condition can include the HAS operator.

Up to seven conditions can be combined using the threshold operators ALL OF and ANY OF to form expressions. Ultimately, a where-clause can start with either threshold operator to combine up to seven expressions.

The qualified data records for a where-clause are always occurrences of the focal record. Each condition refers to a specific schema record: if the condition does not include the HAS operator, the schema record is the record that contains the item specified in the condition; if the condition includes the HAS operator, the schema record is the record specified to the left of HAS. The focal record is the lowest level record of all the records referred to in the conditions in the where-clause.

For a QUEUE language command to be valid, the target record must be related to the focal record.

The following examples show some of the concepts summarized in this section. The data are from the EMPLOYEE data base.

Example 1

The focal record is C200, the target record is C0. Since C0 and C200 are related records, the command is valid.

```
PRINT C2 WHERE C201 EQ COBOL*:
```

Example 2

The focal record is C200, the target record is C0. Since the two records are related, the command is valid.

```
PRINT C2 WHERE ALL OF (C7 EQ FEMALE* , C201 EQ COBOL*):
```

Example 3

The focal record is C120. The target record is C100. The command is valid because the two records are related.

```
PR C101 WH ANY OF (C105 EQ 01/01/1978*, C121 EQ 05/31/1979*):
```

Example 4

The focal record is C200. The target record is C0. The command is valid because the two records are related.

```
PR C2 WH ANY 2 OF (C7 EQ FEMALE*, C201 EQ COBOL*, C201 EQ FORTRAN*):
```

Notice that the previous command is equivalent to the following command:

```
PR C2 WH ALL OF (C7 EQ FEMALE*, ANY OF (C201 EQ COBOL*, C201 EQ FORTRAN*)):
```

Both commands request the last names of those female employees who know at least one of the two languages.

Example 5

C0 is both the target record and the focal record. The command requests the last name of those female employees who know both languages. Notice that the HAS operator was required because of the exclusive conditions.

```
PR C2 WH ALL OF (C7 EQ FEMALE*, C0 HAS C201 EQ COBOL*, C0 HAS C201 EQ FORTRAN*):
```

Example 6

Again, C0 is both the target record and the focal record. Notice that the HAS operator is required because the last two conditions refer to disjoint records.

```
PR C2 WH ALL OF (C7 EQ MALE*, C0 HAS C201 EQ COBOL*, C0 HAS C102 EQ
   INFORMATION SYSTEMS*):
```

Example 7

C200 is both the target record and the focal record. The command retrieves data from those C200 records that contain the value COBOL and are in logical entries for male employees who work or have worked in the Information Systems Department.

```
PR C200 WH ALL OF (C7 EQ MALE*, C201 EQ COBOL*, C0 HAS C102 EQ
   INFORMATION SYSTEMS*):
```

Example 8

The focal record is C0, and the target record is C200; therefore, the command is valid. Notice that the where-clause in this command differs from the where-clause in the Example 7 command in that the second condition includes the HAS operator. This command retrieves data from all C200 records in logical entries for male employees who work or have worked in the Information Systems Department and who know COBOL. For instance, this command retrieves data from records 18, 19, and 47 for David G. Reid (see Data Entry Two at the end of this manual). The command in Example 7, on the other hand, retrieves data only from record 19.

```
PR C200 WH ALL OF (C7 EQ MALE*, C0 HAS C201 EQ COBOL*, C0 HAS C102 EQ
   INFORMATION SYSTEMS*):
```

Example 9

C100 is both the target record and the focal record. Notice that the C100 record is specified in the conditions. Had the C0 record been specified in both conditions, the command might have retrieved C100 records with descendant C110 records that would not have met the criteria. For instance, consider Data Entry Two. The command retrieves data only from record 13. Had the C0 record been specified, the command would have retrieved data from record 16 also.

```
PR C100 WH ALL OF (C100 HAS C111 EQ 1200.00*, C100 HAS C111 EQ 1300.00*):
```

Example 10

C410 is both the target record and the focal record. Thus, the command is valid. This command retrieves C410 (EDUCATION) records that contain a value greater than or equal to 01/01/1975 for item C413 and a value of either MA, MS, MBA, or PHD for item C412 and that are in logical entries having at least two C200 records, one with the value COBOL for C201 and one with the value FORTRAN for C201.

Notice that the command gives the desired result because the last four conditions are exclusive; therefore, when the ANY 4 OF operator is applied to the seven conditions, each of the qualified C410 records must contain one of the C412 values specified as well as meet the other three conditions.

```
PR C410 WH ANY 4 OF (C0 HAS C201 EQ COBOL*, C0 HAS C201 EQ FORTRAN*, C413
   GE 01/01/1975*, (C412 EQ MA*, C412 EQ MS*, C412 EQ MBA*, C412 EQ PHD*):
```

The same results can be obtained with the following command, which has a more complex where-clause.

```
PR C410 WH ALL OF (ALL OF (C0 HAS C201 EQ COBOL*, C0 HAS C201 EQ FORTRAN*),
   C413 GE 01/01/1975*, ANY OF (C412 EQ MA*, C412 EQ MS*, C412 EQ MBS*,
   C412 EQ PHD*)):
```

Chapter 3

Specifying Criteria in Action-Clauses: The If-Clause

INTRODUCTION

This chapter describes the syntax of the if-clause used in action-clauses of non-tree commands. An action-clause allows you to specify the action that is to take place. The action-clause is the portion of the command on the left of the where-clause; if there is no where-clause, it is the entire command. You need to understand the concepts introduced in Chapter 1, "Using the QUEUE Language: Concepts" to understand this chapter.

This chapter also provides you with the background information necessary to formulate meaningful if-clauses. It includes the syntax, considerations, and examples that apply to if-clauses. The various operators used to form if-clauses are also described in this chapter. Chapter 11, "Comparing the QUEST and QUEUE Languages" compares the syntax of the where-clause with the syntax of the if-clause.

THE IF-CLAUSE

If-clauses allow you to specify conditions that items in selected records must meet for an action (or actions) to occur. You can use if-clauses only in non-tree commands. Collect File items are not allowed.

The if-clause specifies criteria to test selected data records. Actions specified after the keyword THEN are performed on those selected data records that satisfy the criteria. If the keyword ELSE is also specified in the command, actions specified after ELSE are performed on those selected data records that do not satisfy the criteria.

- Unary operators (EXISTS, FAILS) verify the existence or nonexistence of values for an item.
- Binary operators (EQ, NE, LE, LT, GE, and GT) compare data items with values or with other data items.
- The threshold operators (ALL OF, ANY OF) combine if-conditions or if-expressions.
- Both key and non-key items can be specified in if-conditions.

An if-clause consists of up to seven *if-expressions* following the keyword IF. Each if-expression, in turn, is made up of no more than seven if-conditions. Each if-condition is a statement of a criterion that a data item must meet. All items specified in if-conditions of an if-clause must belong to the same schema record, that is, the target record for the command.

An if-condition is specified by using any one of the following operators: EXISTS, FAILS, EQ, NE, LE, LT, GE, and GT.

An if-expression is constructed by specifying either one if-condition or by using the threshold operators ANY OF and ALL OF to combine up to seven if-conditions.

An if-clause is constructed by specifying one if-condition, one if-expression, or by using the threshold operators to combine up to seven if-expressions.

The if-clause processes selected data records by accessing one record at a time and checking, for each if-condition, whether the data item to which it applies satisfies the if-condition. If the data items in the record satisfy their respective if-conditions or combination of if-conditions, those actions listed after the keyword THEN are applied to that record. If the action-clause includes the keyword ELSE, those actions listed after ELSE are applied to the record that does not satisfy the if-clause.

Hereafter, the term *if-condition* is used to refer to conditions in an if-clause. The term *condition* is used only in reference to where-clause conditions. Likewise, *if-expressions* refer to expressions in an if-clause. *Expression* still refers to expressions in a where-clause.

By using an if-clause, you tell the software to test values in selected data records. Data in those selected records that pass the test can then be retrieved or updated by the actions specified in the rest of the action-clause. Thus, the if-clause allows you to specify criteria for actions.

In general, criteria for if-clauses can be stated the same way that criteria for where-clauses are stated. The syntax for constructing where-clauses is very similar to the syntax for if-clauses. Generally, all syntax rules that apply to where-clauses also apply to if-clauses. The only exception is that all items specified in if-conditions must belong to the same record, the target record.

The HAS operator is meaningless and invalid in an if-clause. This implies that exclusive if-conditions cannot be preceded by the ALL OF operator or by the ANY OF operator if the threshold is larger than one. Otherwise, the description of where-clause expressions given in **Specifying Expressions with Threshold Operators: ALL OF, ANY OF** on page 2-16 also applies to if-expressions.

The main difference between the syntax of the where-clause and the syntax of the if-clause is that if-conditions provide more options. These options are described below.

Format

```
IF  |if-expression
    |ALL              OF if-expressions
    |ANY [number]
```

```
     if-expression   |if-condition
                     |ALL              OF if-conditions
                     |ANY [number]

 if-condition item   |unaryoperator
                     |binaryoperator value *
                     |*binaryoperator item *
```

number is an integer equal to or greater than one but less than or equal to the number of if-expressions or to the number of if-conditions within the parentheses that follow the ANY OF operator. The default is 1.

item is the item name or component number of an item in the target record. The item can be key or non-key.

unaryoperator EXISTS or FAILS.

binaryoperator EQ, NE, LE, LT, GE, GT, or the following symbols:

Operator	Symbols
EQ	=
GE	>= or => or < or !<
GT	>
LE	<= or =< or > or !>
LT	<
NE	= or !=

value is a literal value, or one of the two system strings *TODAY* or *DATA*. (See *SYSTEM 2000 QUEST Language, Release 11.6 under IBM and CMS* for information on the system strings.)

Considerations

- An if-clause can be used only in non-tree commands; that is, it cannot be used with the APPEND TREE, REMOVE TREE, or PRINT TREE commands.
- Each value specified in an if-clause must have a system separator (the default is the asterisk) immediately following it.
- All if-conditions in an if-clause must refer to items belonging to the target record for the command.
- No more than seven if-expressions or if-conditions can be specified within the parentheses following the keyword OF. Therefore, an if-clause cannot include more than 49 if-conditions.
- The threshold operators ANY OF and ALL OF can be nested only once. For example an if-clause of the following form is invalid.

```
IF ANY OF (if-condition1, ALL OF (if-condition2, if-condition3,
   ANY OF (if-condition4, if-condition5)), if-condition6)
```

- The maximum number of characters allowed for any value specified in an if-clause is 250.

Abbreviations:

EXISTS = EXIST, EXISTING

FAILS = FAIL, FAILING

- The if-clause allows comparison of data items. When this is used in an if-condition, both items must be of the same type, and if the data type is numeric, they must have the same picture.

- Only items for which you have W-authority can appear in an if-clause.

- CALC mode processing does not pertain to if-clauses.

- For the use of if-clauses in non-tree commands see Chapter 7, "Combining Updates and Retrievals."

- Without symbols, the software reads a blank on either side of an operator to separate it from other syntax units. However, with symbols, blanks are not required for separation. Any blanks encountered following a symbol are used as part of the value. For example, these two where-clauses are equivalent:

```
WHERE C1 EQ ABC AND C2 GT DEF:
WHERE C1=ABC&C2>DEF:
```

- The EQ operator and = symbol are not interchangeable in an update action-clause. You must use EQ when specifying values in ADD, CHANGE, ASSIGN, and INSERT commands. However, if the value is a computation, you must use the = symbol.

- If component names contain any of the available symbols, use the component numbers. Also, if you change the system separator to be any of the symbols, syntax errors could occur if you specify that symbol in the where-clause.

Illus. 3.1 **Comparison of If-Clauses and Where-Clauses**

IF-CLAUSES	WHERE-CLAUSES
Can be used in non-tree commands; cannot be used in tree commands.	Required in all QUEUE language retrieval and update commands, except in APPEND TREE C0.
Only items belonging to the target record can be specified.	Items belonging to different records can be specified.
The HAS operator cannot be used.	The HAS operator can be used.
The items specified can be key or non-key.	Only key items can be specified.
The unary operator FAILS can be used, as well as the EXISTS operator.	Only the unary operator EXISTS can be used.
Data items can be compared to a value or to another data item.	Data items can be compared only to a value.

Examples

The examples in this section show the syntax rules for constructing if-clauses. All commands refer to the EMPLOYEE data base shown in the foldout at the end of this manual.

Example 1

The if-clause consists of one if-condition with the binary operator EQ. The item specified in the if-condition (that is, C3) is in the target record (C0).

```
IF C3 EQ RICHARD* THEN PR C1 WH C2 EQ BROWN*:
```

Example 2

The if-clause consists of one if-expression that uses the ALL OF operator. The if-expression contains two if-conditions. Both if-conditions refer to items in the target record C0. The first if-condition uses the unary operator FAILS, the second if-condition uses the binary operator LE.

```
IF ALL OF (C13 FAILS, HIRE DATE LE 01/01/76*) THEN PR C1
   WHERE C102 EQ MARKETING*:
```

Example 3

The if-clause consists of one if-expression that uses the ANY OF operator. The if-expression contains three if-conditions. All three of these refer to items in the target record (C100).

```
IF ANY OF (C101 EQ PROGRAMMING*, C106 FAILS, C105 GE 01/01/77*)
   THEN PR C102 WH C1 EQ 1120*:
```

Example 4

The if-clause consists of the ALL OF operator followed by three if-expressions. The first one is an if-condition. The second if-expression uses the ALL OF operator and includes two if-conditions. The last one uses the ANY OF operator and includes two if-conditions. All items in the if-clause belong to the target record C0.

```
IF ALL OF (C13 FAILS, ALL OF (C4 GE 01/01/76*, C4 LE 12/31/77*),
   ANY OF (C16 GE 78701*, C16 LE 78769*))
   THEN PR C0 WH C102 EQ MARKETING*:
```

Example 5

The if-clause consists of the ALL OF operator followed by three if-expressions. The first two if-expressions are if-conditions. The second one compares two data items. The third if-expression uses the ANY 2 OF operator to combine three conditions. All items in the if-clause belong to the target record C0.

```
IF ALL OF (C13 FAILS, C11* EQ C12*, ANY 2 OF (C15 EQ AUSTIN TX*,
   C16 GE 78700*, C16 LE 78799*)) THEN PR C0 ELSE PR C1, PR C8 WH
   C102 EQ MARKETING*:
```

Other examples showing if-clause processing can be found in **Examples of Non-Tree Commands** on page 7-2.

Examples showing the use of if-clauses are found in Chapter 11, "Comparing the QUEST and QUEUE Languages."

Examples showing the syntax rules for the various operators in the if-clause can be found in **Specifying If-Conditions** on page 3-8 through **Comparing Two Data Items: EQ, NE, GE, GT, LE, or LT** on page 3-14.

SPECIFYING IF-CONDITIONS

An if-expression consists of up to seven if-conditions, which you specify with the operators described in this section. There are four general types of if-conditions; they are described in the next few pages in the order given here:

1. specifying if-conditions that test for the existence of values using EXISTS.
2. specifying if-conditions that test for the nonexistence of values using FAILS.
3. specifying if-conditions that compare a data item with a value using EQ, NE, LE, LT, GE, or GT.
4. specifying if-conditions that compare two data items using EQ, NE, LE, LT, GE, or GT.

Uses of the If-Clause

An if-clause can be used for one or more of the following reasons:

- Because the syntax for if-conditions provides more flexibility than the syntax for conditions, the if-clause can specify criteria that cannot be stated in the where-clause. For example, the following command is invalid because C15 is a non-key item, so it cannot be specified in a condition.

```
PRINT C2 WHERE ALL OF (C4 GE 01/01/1977*, C15 EQ AUSTIN TX*):
```

  However, the request can be satisfied by issuing this command:

```
IF C15 EQ AUSTIN TX* THEN PRINT C2 WHERE C4 GE 01/01/1977*:
```

- Because non-tree commands allow the format IF-THEN-ELSE, the if-clause can be used to state criteria that require alternate action(s) when they are not met. For example, a request to retrieve the names and office-extensions of employees who live in Austin and to retrieve the name and address of those employees who live elsewhere would require two separate commands or one command that would retrieve the required data for every employee. However, by using the IF-THEN-ELSE format, the information can be obtained with just one command and without retrieving unnecessary data.

```
IF C15 EQ AUSTIN TX* THEN PRINT C2, PRINT C3, PRINT C10
   ELSE PRINT C2, PRINT C3, PRINT C14, PRINT C15, PRINT C16
   WHERE C1 EXISTS:
```

- Because if-conditions are applied only to items belonging to records that have already been selected, criteria that could be stated in the where-clause can, instead, be specified in the if-clause for efficiency . For example, assume that the EMPLOYEE data base contains data on 10,000 employees. Of these, five were born on January 1, 1940, and 9,500 are full-time employees. There are basically two QUEUE language commands that can be used to identify the full-time employees born on that date.

```
PRINT C1 WHERE ALL OF (C9 EQ FULL TIME*,C5 EQ 01/01/1940*):

IF C9 EQ FULL TIME THEN PRINT C1 WHERE C5 EQ 01/01/1940*:
```

To execute the first command, the software must first find the Hierarchical Table addresses of those 9,500 C0 records where C9 EQ FULL TIME and the addresses of those five C0 records where C5 equals 01/01/1940. Then it must compare the two lists to find which of those C0 records contain the two values. Assume there are three such records. Those are the qualified data records and, in this case, also the selected data records. The selected data records are then arranged by Hierarchical Table address, and the value for C1 obtained from each record.

To execute the second command, the software finds the Hierarchical Table addresses of the five C0 records with the required value for C5. These are the qualified data records and also the selected data records.

These five records are arranged according to their address and accessed one at a time. As each record is accessed, the software checks to see whether the record contains the value of FULL TIME for item C9. If so, the software retrieves the value for C1 and prints it; if not, the software discards the record.

Therefore, when possible, it is more efficient to place conditions that qualify large numbers of records in the if-clause. This consideration, however, applies only to non-tree commands because the if-clause is not allowed in tree commands.

THE EXISTS OPERATOR

The unary operator EXISTS tests for the existence of values. EXISTS is called a unary operator because it has only one operand: an item. EXISTS qualifies all selected data records that have a value for the specified item.

Only items in the target record can be specified with the EXISTS operator.

Format

item **EXISTS**

item is the name or component number of an item in the target record for the command.

Considerations

Care must be taken when an if-condition on an item uses the EXISTS operator to state criteria for update actions on the same item. For example, consider the following two commands:

```
IF C13 EXISTS THEN ADD C13 EQ 121* WH C102 EQ MARKETING*:

IF C13 EXISTS THEN CHANGE C13 EQ 121* WH C102 EQ MARKETING*:
```

The update action in the first command would not occur, because ADD acts only on items without values; that is, it enters values where none exist. Therefore, this command is contradictory. (See **Replacing a Null with a Value: ADD** on page 5-3.)

The if-clause in the second command is superfluous, because CHANGE acts only on items with values. Therefore, the second command is equivalent to this command:

```
CHANGE C13 EQ 121* WH C102 EQ MARKETING*:
```

Example

The if-condition in the following command identifies those selected C0 data records that have a value for C15. If a record does contain a value, then values for C1, C14, C15, and C16 are retrieved.

```
IF C15 EXISTS THEN PR C1, PR C14, PR C15, PR C16 WH C102 EQ MARKETING*:
```

Abbreviation: EXISTS = EXIST, EXISTING

THE FAILS OPERATOR

The unary operator FAILS tests for the nonexistence of values. FAILS is called a unary operator because it has only a single operand: an item. EXISTS qualifies all selected data records that do not have a value for an item.

Only items in the target record can be specified with the FAILS operator.

Format

item **FAILS**

item is the name or component number of an item in the target record for the command.

Considerations

Care must be taken when an if-condition on an item uses the FAILS operator to state criteria for update actions on the same item. For example, consider the following two commands:

```
IF C13 FAILS THEN CHANGE C13 EQ 121* WH C102 EQ MARKETING*:

IF C13 FAILS THEN ADD C13 EQ 121* WH C102 EQ MARKETING*:
```

The update action in the first command would not occur because CHANGE acts only on items with values, that is, it changes values where they already exist. Therefore, the command is contradictory. The if-clause in the second command is superfluous, because ADD acts only on items without values. Therefore, the second command is equivalent to the following command:

```
ADD C13 EQ 121* WH C102 EQ MARKETING*:
```

Example

The if-condition in the following command identifies those selected C0 data records that do not have a value for C13. If a record does not contain a value, the value for C4 is retrieved.

```
IF C13 FAILS THEN PR C4 WH C102 EQ MARKETING*:
```

Abbreviations: FAILS = FAIL, FAILING

COMPARING A DATA ITEM WITH A VALUE: EQ, NE, LE, LT, GE, OR GT

The binary operators EQ, NE, LE, LT, GE, and GT (or their equivalent symbols) compare data items in selected data records with a given value. Binary operators are so called because they have two operands: an item and a specified value.

EQ qualifies those selected data records that contain the specified value for the item. (CALC mode processing is used for a condition containing a CALC item with the EQ operator.) For example, in this command, the if-condition identifies those selected C0 data records that contain the value MALE for item C7.

```
IF C7 EQ MALE* THEN PR C1, PR C2 WH C102 EQ MARKETING*:
```

NE qualifies those selected data records containing a value for the item that is not equal to the specified value. For example, in this command, the if-clause identifies those selected C0 data records that contain a value for C8 that is not equal to CAUCASIAN. It does not qualify those records where the value for C8 is null.

```
IF C8 NE CAUCASIAN* THEN PR C8 WH C102 EQ MARKETING*:
```

GE qualifies those selected data records containing a value for the item that is greater than or equal to the specified value. For example, in this command, the if-clause identifies those C0 selected data records that contain a value for C4 that is greater than or equal to 01/01/77*, that is, a date after or on January 1, 1977.

```
IF C4 GE 01/01/87* THEN PR C1 WH C102 EQ MARKETING*:
```

GT qualifies those selected data records containing a value for the item that is greater than the specified value. For example, in the following command, the if-clause identifies those C0 selected data records containing a value for C4 that is greater than 01/01/87*, that is, a date after January 1, 1987.

```
IF C4 GT 01/01/87* THEN PR C1 WH C102 EQ MARKETING*:
```

LE qualifies those selected data records containing a value for the item that is less than or equal to the specified value. For example, in this command, the if-clause identifies those C0 selected data records containing a value for C4 that is less than or equal to 01/01/87*, that is, a date before or on January 1, 1987.

```
IF C4 LE 01/01/87* THEN PR C1 WH C102 EQ MARKETING*:
```

LT qualifies those selected data records containing a value for the item that is less than the specified value. For example, in this command, the if-clause identifies those C0 selected data records containing a value for C4 that is less than 01/01/87*, that is, a date before January 1, 1987.

```
IF C4 LT 01/01/87* THEN PR C1 WH C102 EQ MARKETING*:
```

In this command, the if-clause identifies those C0 selected data records containing a value for C3 that is less than B, that is, those employees, who work or have worked in the Marketing Department and whose first names start with the letter A.

```
IF C3 LT B* THEN PR C2, PR C3 WH C102 EQ MARKETING*:
```

Format

```
item |EQ value systemseparator
     |NE
     |GE
     |GT
     |LE
     |LT
```

item is the name or component number or a key or non-key item in the target record for the command.

value is a literal value or one of the two system strings *DATA* or *TODAY*. (See *SYSTEM 2000 QUEST Language, Release 11.6 under IBM and CMS* for more information on system strings.)

systemseparator is a single, special character. The default is the asterisk. (See *SYSTEM 2000 QUEST Language* for a list of system separators.)

Considerations

- An asterisk (the default system separator) always follows the specified value. If *DATA* is used, no asterisk is specified after it. For example, in this if-condition, the asterisk is required.

  ```
  C2 EQ BROWN*
  ```

  However, in this if-condition the use of another asterisk is invalid.

  ```
  C2 EQ *DATA*
  ```

- When a value is specified for a TEXT item, the blank space after the binary operator acts as a delimiter for the operator and is not considered part of the value. However, all blanks after the first and before the asterisk are considered part of the value. For example, this if-condition is satisfied by those selected C130 records that have the value ƀABCƀƀ for item C132.

  ```
  C32 EQƀƀABCƀƀ*
  ```

COMPARING TWO DATA ITEMS: EQ, NE, GE, GT, LE, OR LT

The binary operators EQ, NE, GE, GT, LE, and LT (or their equivalent symbols) compare a data item with a value. These operators can also be used to compare two data items. When used for comparing two data items, binary operators have two operands: the two items to be compared. The comparison between the two data items is based on the same principles as the ones described when the operators are used to compare an item with a value. For example, the EQ operator qualifies selected data records where the first data item has the same value as the second data item, the LT operator qualifies selected data records where the value of the first data item is less than the value of the second data item, and so on.

To be eligible for comparison, the data items must meet the following requirements:

- Both items must be in the target record.
- Both items must be of the same type, for example, both TEXT or both INTEGER.
- If they are numeric, their pictures must be equal. If they are CHARACTER or TEXT, the pictures can be different.

If either item is null for a particular selected data record, the software does not compare the items, and the record cannot satisfy the if-condition.

CALC mode processing is not used in item-to-item comparison. (CALC mode processing is explained in *SYSTEM 2000 QUEST Language*.)

Format

```
item systemseparator |EQ item systemseparator
                     |NE
                     |GE
                     |GT
                     |LE
                     |LT
```

item is the name or component number of a key or non-key item in the target record.

systemseparator is a single, special character. The default is the asterisk. (See *SYSTEM 2000 QUEST Language* for a list of system separators.)

Examples

Example 1

In this command, the if-clause identifies those C0 selected data records containing the same value for C11 and for C12. Notice that the comparison is possible because both items are DECIMAL and the picture for both is 999.99.

```
IF C11* EQ C12* THEN PR C1, PR C11 WH C102 EQ MARKETING*:
```

Example 2

The if-clause identifies those C100 selected data records (in this case, 13 and 16) containing a value for C106 that is greater than the value for C105. Notice that the comparison is possible because both items are dates.

The first selected record to be tested is 13. In record 13, there is no value for C106; therefore, the values are not compared and the action listed after ELSE occurs; that is, the value for C102 is retrieved.

The second record to be tested is 16. In that record, the value for C106 is 02/28/1977, and the value for C105 is 02/16/1976; therefore, the if-condition is satisfied, and the value for C101 is retrieved.

The following command refers to Data Entry Two in the foldout at the end of this manual.

```
IF C106* GT C105* THEN PR C101 ELSE PR C102 WH C1 EQ 1120*:
```

Chapter 4

Retrieving Data

INTRODUCTION

This chapter describes the syntax of QUEUE retrieval commands. You need to understand the concepts introduced in Chapter 1, "Using the QUEUE Language: Concepts" in order to understand this chapter.

In the QUEUE language, one tree command (PRINT TREE) and one non-tree action (PRINT) retrieve data from a SYSTEM 2000 data base.

Format options can be set with any one of the five QUEST language commands (LIST, PRINT, UNLOAD, COLLECT, or FORMOP) before entering a QUEUE/TERMINATE session. Five of the eight format options can affect the output of QUEUE language retrievals. They are: SINGLE SPACE/DOUBLE SPACE, INDENT/BLOCK, STUB/STUB SUPPRESS, NUMBER/NAME, and NULL SUPPRESS/NULL. (See *SYSTEM 2000 QUEST Language, Release 11.6 under IBM and CMS* for details on format options.)

Issuing the TERMINATE command with the UNLOAD option (that is, TERMINATE/UNLOAD) causes the output of all retrievals issued within that QUEUE/TERMINATE session to be written in UNLOAD format. (See *SYSTEM 2000 QUEST Language* for details.)

RETRIEVALS FROM DATA RECORDS: PRINT

The purpose of a PRINT action in a non-tree command is to retrieve values for all items in selected data records or to retrieve values for only specified items in selected data records.

A PRINT action

- displays all values in selected data records
- displays values for specified items in selected data records

- displays data in physical order
- must be used in a non-tree command
- can be specified after THEN or ELSE following an if-clause
- can be specified several times within a non-tree action-clause
- can be specified with other non-tree actions.

PRINT displays data in a simple sequential list, one value to a line.

When an item in the target record is specified after PRINT, the software displays values for that item in selected data records. When the target record is specified after PRINT, the software displays the values in selected data records. Data from descendant records are not displayed. The values are always displayed in physical order, that is, by the Hierarchical Table address of the data record to which they belong.

Several PRINT actions can be specified in the action-clause of one non-tree command, see **Action-Clauses in Non-Tree Commands: IF-THEN-ELSE** on page 7-5 for details.

A PRINT action can be specified in any non-tree command. Only the format of PRINT is given here. Refer to Chapter 7, "Combining Updates and Retrievals" for the complete format of non-tree commands.

Format

```
PRINT |record
      |item
```

record is the name or component number of the target record.

item is the name or component number of an item in the target record.

Examples

The examples in this section show the syntax of PRINT actions. Other examples are shown in **Retrievals from Data Trees: PRINT TREE** on page 4-5. The following examples show non-tree commands that include PRINT actions. All commands refer to Data Entry Two in the foldout at the end of this manual. Each command is followed by its resulting output.

Abbreviation: PRINT = PR

Example 1

This command includes one PRINT action specifying an item.

```
PRINT C201 WHERE C1 EQ 1120*:

201* PL/1
201* COBOL
201* FORTRAN
```

Example 2

This command includes a PRINT action specifying a record.

```
PR C200 WHERE C1 EQ 1120*:

201* PL/1
202* EXCELLENT
203*   5

201* COBOL
202* GOOD
203*   1

201* FORTRAN
202* GOOD
203*   1
```

Example 3

In this command, the first PRINT action identifies an item by its component number, and the second PRINT action identifies an item by its name.

```
IF C203 EQ 1* THEN PR C201 ELSE PRINT PROFICIENCY
   WHERE C1 EQ 1120*:

202* EXCELLENT
201* COBOL
201* FORTRAN
```

Example 4

In this example, the QUEST language LIST command is used to specify the NULL format option. This causes nulls to be displayed in the output of retrieval commands. The PRINT action in the non-tree command specifies a record.

```
LIST /NULL/:
---
QUEUE:
---
IF C101 EQ PROGRAMMER* THEN PR C100 WH C1 EQ 1120*:
---
T:

--SEQUENCE NUMBER--1

101* PROGRAMMER
102* INFORMATION SYSTEMS
103* MYJ
104* PROFESSIONAL
105* 03/01/1977
106* -NULL-
```

RETRIEVALS FROM DATA TREES: PRINT TREE

The purpose of the PRINT TREE command is to retrieve data from selected data records and their descendants. The PRINT TREE command

- can include only one record in the action-clause
- displays values in data trees in logical order
- must be used with a where-clause.

The PRINT TREE command displays data in a simple sequential list, one value to a line. Only one record can be specified after PRINT TREE. This is the target record for the command. Selected records are those occurrences of the target record that are related to the qualified data records. The output always includes the selected data records and their descendant records. The values from each data tree are displayed in logical order.

As with retrieval commands in the QUEST language, you can specify format options to change the appearance of the output from PRINT TREE and PRINT. However, format options can only be specified with QUEST language commands: LIST, PRINT, UNLOAD, COLLECT, or FORMOP. Therefore, you must specify a format option for QUEUE retrievals before beginning a QUEUE/TERMINATE session.

You also have the option of specifying TERMINATE/UNLOAD at the end of a session. As a result, all the data retrieved by PRINT TREE and PRINT during that QUEUE/TERMINATE session is written to the Report File in a format suitable for input to other software commands. (See the discussion for UNLOAD in *SYSTEM 2000 QUEST Language* for details on this format.)

For a retrieval action to be processed by the software, you must have R-authority for the item or record to the right of PRINT and for the record to the right of PRINT TREE. You must have W-authority for any component in the where-clause of the command. If the command includes an if-clause, W-authority is also required for every item specified in it.

Format

PRINT TREE *record where-clause;*

record is a record name or component number. The record specified after PRINT TREE must be related to the focal record for the where-clause.

where-clause See Chapter 2, "Specifying Qualification Criteria: The Where-Clause." A where-clause must always be included.

Abbreviation: PRINT = PR

Examples

All commands in these examples refer to Data Entry Two in the foldout at the end of this manual.

Example 1

The where-clause in the following command qualifies the C0 record. Because the target record is also C0, the command displays all of the values in the logical entry in order.

```
PRINT TREE C0 WH C1 EQ 1120*:
```

Example 2

The where-clause in the following command qualifies data record 14.

```
PR TREE C110 WH ALL OF (C1 EQ 1120*, C113 EQ 03/01/78*):


111*   $1,200.00
112* MONTHLY
113* 03/01/1978
114*     $215.40

  121* 02/28/1979
  122*  160.00
  123*     .00
  124*   $1,200.00
  125*     $135.60
  126*     $984.60
```

Example 3

The command in Example 2 displays the item number prior to the corresponding value. The NAME format option can be set prior to entering the QUEUE/TERMINATE session so that item names appear before the values. The command and output are shown below.

```
LIST /NAME/:
---
QUEUE:
---
PRINT TREE C110 WH ALL OF (C1 EQ 1120*, C113 EQ 03/01/78*);
---
TERMINATE:


--SEQUENCE NUMBER--1

   PAY RATE*   $1,200.00
   PAY SCHEDULE* MONTHLY
   EFFECTIVE DATE* 03/01/1978
   CURRENT DEDUCTION*     $215.40

     PAYROLL MONTH* 02/28/1979
     REGULAR HOURS*  160.00
     OVERTIME HOURS*     .00
     GROSS PAY*   $1,200.00
     FEDERAL TAX DEDUCTION*     $135.60
     NET PAY*     $984.60

     PAYROLL MONTH* 01/31/1979
     REGULAR HOURS*  184.00
     OVERTIME HOURS*   12.00
     GROSS PAY*   $1,200.00
     FEDERAL TAX DEDUCTION*     $135.60
     NET PAY*     $984.60
---
```

Example 4

The following command retrieves the values in the logical entry from the employee where EMPLOYEE NUMBER (C1) equals 1043. The records are displayed in logical order.

```
PRINT TREE C0 WHERE C1 EQ 1043*:
```

The following non-tree command retrieves all values in the C0 record of the same logical entry.

```
PRINT C0 WHERE C1 EQ 1043*:
```

Finally this command retrieves only the LAST NAME (C2) of that employee.

```
PRINT C2 WHERE C1 EQ 1043*:
```

As discussed in **Multiple Actions** on page 7-7, several PRINT actions can be included in one non-tree command. For example, the following command retrieves the LAST NAME (C2) and FORENAME (C3) of the same employee.

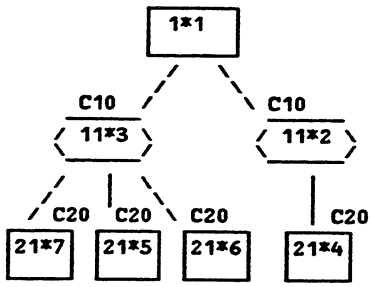
```
PRINT C2, PRINT C3 WHERE C1 EQ 1043*:
```

Both PRINT TREE and PRINT retrieve values in a simple sequential list, double spacing before data records. Collect File items are not allowed.

PROCESSING

The processing of retrieval commands are shown in the following examples. The first four examples use the data base shown below. Examples 5 and 6 refer to Data Entry Two in the foldout at the end of the manual.

In the data tree shown below the values for the items in each record correspond to the Hierarchical Table address of each record.



Example 1

In this example, the focal record is C0, the target record is C10; therefore, the command is valid. The qualified data record is the C0 record. The selected data records are the descendant C10 records with C11 values 3 and 2. The records in the data tree topped by the record with value 3 are arranged in logical order, and so are the records in the second data tree. The data are then retrieved and displayed in that order, as shown below.

```
PRINT TREE C10 WHERE C1 EXISTS:


11*    3
  21*      7
  21*      5
  21*      6
11*    2
  21*      4
```

Example 2

In this example, the focal record and the target record are both C10. The qualified data records have C11 values 2 and 3; these become the selected data records. As in Example 1, the data are displayed in logical order; however, because there were two qualified data records and these were sorted, the data tree topped by the C10 record with C11 value 2 is displayed before the other data tree.

```
PRINT TREE C10 WHERE C11 EXISTS:


  11*    2
    21*      4
  11*    3
    21*      7
    21*      5
    21*      6
```

Example 3

In this example, the focal record is C10, and the target record is C20; therefore, the command is valid. The qualified data records are C10 records with C11 values 2 and 3. The selected data records are those C20 records that are descendants of the qualified data records. The descendant of the C10 record with C11 value 2 is the C20 record with C21 value 4. The descendants of the C10 record with C11 value 3, in logical order, are C20 records with values 7, 5, and 6. That order is used when the records are displayed.

```
PRINT TREE C20 WHERE C11 EXISTS:


  21*    4
  21*    7
  21*    5
  21*    6
```

Example 4

In this example, C20 is both the focal record and the target record. The qualified data records are C20 records with values 4, 5, 6, and 7. The selected records are the qualified data records. The command displays the records in that order.

```
PRINT TREE C20 WHERE C21 EXISTS:


 21*    4
 21*    5
 21*    6
 21*    7
```

The previous four examples show that when the focal record is above the level of the target record, the data records at the top of the data trees appear in logical order, but when the focal record is at or below the level of the target record, the data records at the top of the data trees appear in physical order.

Example 5

In this example, the focal record is C0, and the target record is C110; therefore, the command is valid. The selected data records are those C110 records that are descendants of the qualified data record. Before the data are retrieved, the selected data records are sorted by Hierarchical Table address: 14, 15, 17, and 38. The values from each record are then displayed in that order. The output is shown below.

```
PRINT C110 WHERE C1 EQ 1120*:


111*   $1,200.00  ------
112* MONTHLY             \   from record 14
113* 03/01/1978          /
114*     $215.40  ------

111*   $1,100.00   ------
112* MONTHLY             \   from record 15
113* 03/01/1977          /
114*     $193.14   ------

111*   $1,000.00   ------
112* MONTHLY             \   from record 17
113* 02/16/1976          /
114*     $171.27   ------

111*   $1,234.57  ------
112* MONTHLY             \   from record 38
113* 03/01/1979          /
114*     $239.65  ------
```

Example 6

The same considerations that apply to the command in Example 5 also apply to this one. However, only the values for C111 and C114 are displayed here as shown below:

```
PRINT C111, PRINT C114 WHERE C1 EQ 1120*:


111*   $1,200.00 \
114*     $215.40 /  from record 14
111*   $1,100.00 \
114*     $193.14 /  from record 15
111*   $1,000.00 \
114*     $171.27 /  from record 17
111*   $1,234.57 \
114*     $239.65 /  from record 38
```

Chapter 5

Updating Data Records

INTRODUCTION

This chapter describes the syntax of the four update actions in non-tree commands. You need to understand the concepts in Chapter 1, "Using the QUEUE Language: Concepts" in order to understand this chapter.

The four actions used to update data items in selected data records are as follows:

- ADD replaces null items with values.
- CHANGE replaces values with new values.
- ASSIGN replaces nulls or valued items with new values.
- REMOVE replaces values with nulls.

These actions affect only existing data records; they do not create or delete data records. In other words, they do not modify the structure of a data base. To modify the structure of a data base, you need to use the tree commands described in Chapter 7, "Combining Updates and Retrievals."

The data base cycle number is incremented once for each QUEUE/TERMINATE session including one or more update commands that are successfully executed. This number is displayed in the output of the QUEST language DESCRIBE command.

OUTPUT FROM UPDATE COMMANDS

Update commands do not produce output similar to the output resulting from retrieval commands. Only two types of messages can occur as a result of an update command:

- An error message when there is an error in the command. The message appears immediately after the command.
- -373- DATA BASE UNALTERED - message. This message appears immediately after the TERMINATE command when all the executed update commands within the session cause no change to the data base. For example, if the following update command were the only one in a session, message 373 would appear after the TERMINATE command.

```
IF C4 FAILS THEN REMOVE C4 WHERE C1 EXISTS:
```

  If you desire verification of the results of an update command, you can issue appropriate retrieval commands either in conjunction with the update action or after the update action. For example, this command displays the values for C3 both before and after the change. You can, of course, issue separate commands for each action.

```
PRINT C3, CHANGE C3 EQ JOHN*, PRINT C3 WHERE C2 EQ BROWN*:
```

  The data base is marked internally if any update process completes abnormally, as in such cases as fatal system errors, time limit job abends, operator drops, system drops, and so on. (Syntax errors do not cause damage to a data base.)

Subsequently, if you try to perform an update on a data base damaged from a previous job, the following message is issued.

```
-810- WARNING- DATA BASE DAMAGED. NO UPDATES PERMITTED -
```

This message appears on the Message File of every job that uses a data base damaged by a previous job abend. The message is also printed if you attempt to use an update command. The data base can still be saved, restored, and described, and retrievals are allowed.

FORMAT SPECIFICATIONS AND OTHER SYNTAX CONSIDERATIONS

Format specifications for the ADD, CHANGE, and ASSIGN actions are identical except for the keywords ADD, CHANGE, and ASSIGN. The REMOVE action uses a different format. Since it is used only to remove values, there is no need to specify new values.

In ADD, CHANGE, and ASSIGN actions, component numbers and values to the right of the EQ operator must each be followed by a system separator, for example, 1*1120*2*REID*.

When a new value is specified for an item in an update action, the value must be coded to conform to the type and picture of the item. The discussion on coding values given in *SYSTEM 2000 QUEST Language, Release 11.6 under IBM and CMS* also applies in this case. The only difference is in coding TEXT values. In this case, the blank space after the EQ operator acts as a delimiter for the operator and does not become part of the value. For example, this command adds the value ABC to item C132.

```
ADD C132 EQ␣ABC* WHERE C1 EQ 2345*;
```

However, this command adds the value ␣ABC to that item.

```
ADD C132 EQ␣␣ABC* WHERE C1 EQ 2345*;
```

REPLACING A NULL WITH A VALUE: ADD

The purpose of an ADD action in a non-tree command is to replace nulls with values. An ADD action

- affects only selected data records
- replaces one or more null items with values.

An ADD action adds values only where the specified items have no values. Thus, you can use this action and let the software check the contents of each selected data record for the existence of values. An ADD action affects only selected data records, that is, those occurrences of the target record that are related to the qualified data records. There are two acceptable formats for an ADD action. Use format 1 to add a value to a single item; use format 2 to add values to multiple items.

With format 1, the software checks each selected data record to see whether the item is valued. If so, it replaces that value with the value specified in the action; if the item is already valued, no action takes place.

With format 2, the software checks each item specified in the value stream to see if it is currently valued in the selected data record being acted upon. Items found to be null have values added to them; items found to be valued are unaffected. If an item is not specified in the value stream, it is not affected. The items in the value stream can be given in any order.

An ADD action can be specified in any non-tree command. Only the format of ADD is given here. Refer to **Action-Clauses in Non-Tree Commands: IF-THEN-ELSE** on page 7-5 for the complete format of non-tree commands.

Format

```
(1) ADD item  EQ   |value*
                   |*DATA*

(2) ADD record EQ  |valuestream END*
                   |*DATA*
```

item is the name or component number of an item in the target record.

value is a literal value, or one of the two systems strings *TODAY* or *NOW* (See *SYSTEM 2000 QUEST Language* for information on system strings.)

record is the name or component number of the target record.

valuestream is a series of item numbers, each followed by its associated value to be added. The system separator must follow each item number and each value. The item numbers need not be in DESCRIBE output sequence. All items must belong to the target record. The default system separator is the asterisk.

DATA see Chapter 8, "Repetitive Processing."

Considerations

- An ADD action can be specified only in a non-tree command.
- When format 1 is used, the item specified must belong to the target record.
- When format 2 is used, the record specified must be the target record and the items in the value stream must belong to that record.

Abbreviation: ADD = AD

- An ADD action adds values only when the specified item (items) is (are) not currently valued.
- If a non-tree command including an ADD action is issued for a damaged data base, the command is not executed.
- Only items and records for which you have U-authority can be specified in an ADD action.
- The ADD action cannot be used for a CALC item because CALC items always have an existing value (used to locate the CALC record). See *SYSTEM 2000 QUEST Language* for details on CALC.
- If you use format 2 without specifying a value stream, no update occurs.
- The QUEST language ADD command can also be used to enter new values in existing data records.

Examples

The commands in the following examples refer to the EMPLOYEE data base.

Example 1

The commands in this example use format 1.

```
ADD SECURITY CLEARANCE EQ 106* WH C1 EQ 1043*:

ADD C121 EQ 05/31/1979* WH C102 EQ MARKETING*:

AD C16 EQ 78752* WH C1 EQ 1120*:
```

Example 2

For the effects of this command on one selected data record, see *SYSTEM 2000 QUEST Language*.

```
ADD C412 EQ BS* WH C1 EQ 1005*:
```

Example 3

The commands in this example use format 2.

```
ADD EDUCATION EQ 413* 01/07/73* 414* ENGLISH* 415* MARKETING*
   END* WHERE C2 EQ BROOKS*:

AD C0 EQ 11*40.00* 12* 56.00* END* WH C1 EQ 1043*:
```

Example 4

For the effects of this command on one selected data record, see *SYSTEM 2000 QUEST Language*.

```
AD C410 EQ 412*BA* 414*MATH* 415* ECON* END* WH C1 EQ 1005*:
```

REPLACING A VALUE WITH A NEW VALUE: CHANGE

The purpose of a CHANGE action in a non-tree command is to replace values with new values. A CHANGE action

- affects only selected data records
- replaces one or more values with corresponding new values.

A CHANGE action changes values only where the specified items have values. Also, it affects only selected data records, that is, those occurrences of the target record that are related to the qualified data records.

There are two formats for a CHANGE action. If you are changing a value for a single item, use format 1. If you are changing values for multiple items, use format 2.

With format 1, the software checks each selected data record to see whether the item is valued. If so, it replaces that value with the value specified in the action; if the item is null, no action occurs.

With format 2, the software checks each item specified in the value stream to see if it is currently valued in the selected data record being acted upon. Items found to be valued are replaced by the new specified value; items found to be null are unaffected.

If an item is not specified in the value stream, it is not affected. The items in the value stream can be given in any order.

A CHANGE action can be specified in any non-tree command. Only the format of CHANGE is given here. Refer to **Action-Clauses in Non-Tree Commands: IF-THEN-ELSE** on page 7-5 for the complete format of non-tree commands.

Format

```
(1) CHANGE item EQ    |value*
                      |*DATA*

(2) CHANGE record EQ  |valuestream END*
                      |*DATA*
```

item is the name or component number of an item in the target record.

value is a literal value, or one of the two system strings *TODAY* or *NOW*. See *SYSTEM 2000 QUEST Language*.

record is the name or component number of the target record.

Abbreviation: CHANGE = CH

valuestream is a series of item numbers, each followed by its associated value to be added. The system separator must follow each item number and each value. The item numbers need not be in DESCRIBE output sequence. All items must belong to the target record. The default system separator is the asterisk.

DATA see Chapter 8, "Repetitive Processing."

Considerations

- A CHANGE action can be specified only in a non-tree command.
- When format 1 is used, the item specified must belong to the target record.
- When format 2 is used, the record specified must be the target record and the items in the value stream must belong to that record.
- A CHANGE action adds values only when the specified item (items) is (are) currently valued.
- If a non-tree command including a CHANGE action is issued for a damaged data base, the command is not executed.
- Only items and records for which you have U-authority can be specified in a CHANGE action.
- CALC item values cannot be changed if the UNIQUE option is in effect.
- If format 2 is used without specifying a value stream, no update occurs.
- The QUEST language CHANGE command can also be used to change values in existing data records.

Examples

The commands in the following examples refer to the EMPLOYEE data base.

Example 1

The commands in this example use format 1.

```
CHANGE PAY SCHEDULE EQ MONTHLY* WHERE C112 EQ MONTHLY*:

CH C11 EQ 80.00* WH C1 EQ 1120*:

IF C121 EQ 05/29/79* THEN CH C126 EQ 1060.35* WH C1 EQ 1120*:
```

Example 2

For the effects of this command on one selected data record, see *SYSTEM 2000 QUEST Language*.

```
CHANGE C412 EQ BA* WHERE C1 EQ 1005*:
```

Example 3

The commands in this example use format 2.

```
CHANGE JOB SKILLS EQ 202* GOOD* 203* 3* END*
   WHERE ALL OF (C201 EQ GRAPHICS*, C1 EQ 1043*):

CH C0 EQ 14* 1302 LAZY LANE* 16* 78752* END*
   WHERE C1 EQ 1120*:
```

Example 4

For the effects of this command on one selected data record, see *SYSTEM 2000 QUEST Language*.

```
CHANGE C410 EQ 412* BS* 414* MATHEMATICS* 415* ECONOMICS* END*
   WHERE C1 EQ 1005*:
```

REPLACING A VALUE OR A NULL WITH A VALUE: ASSIGN

The ASSIGN action in a non-tree command assigns values to items regardless of whether the items are valued. An ASSIGN action

- affects only selected data records
- replaces one or more items with values.

An ASSIGN action alters the contents of selected data records by replacing values or nulls with the new values specified. Unlike ADD or CHANGE actions, which are conditional upon the status of the values for the items, ASSIGN is unconditional; it always assigns the new value, provided the item does not already contain the new value. An ASSIGN action affects only selected data records, that is, those occurrences of the target record that are related to the qualified data records. There are two formats for an ASSIGN action. Use format 1 when you are assigning a value to a single item; use format 2 when you are assigning new values to multiple items.

With format 1, the software checks each selected data record to see whether the item is valued. If so, no action takes place; otherwise, any other value, if it exists, is removed and replaced by the new value.

With format 2, the software checks each item in the selected data record being acted upon. If the current value for an item is the same as the new value specified for that item in the value stream, the item is not affected. Otherwise, any other value, if it exists, is removed and replaced by the new value specified. If no value is specified for an item in the value stream, the item becomes null. Therefore, when format 2 is used and any of the original values are still to be retained, the value stream must include the original values or that data will be lost in the removal. In other words, data for items not specified in the value stream are removed from the data base. Items in the values stream can be given in any order.

An ASSIGN action can be specified in any non-tree command. Only the format of ASSIGN is given here. Refer to **Action-Clauses in Non-Tree Commands: IF-THEN-ELSE** on page 7-5 for the complete format of non-tree commands.

Format

```
(1) ASSIGN item EQ      |value*
                        |*DATA*

(2) ASSIGN record EQ    |valuestream END*
                        |*DATA*
```

item is the name or component number of an item in the target record.

Abbreviation: ASSIGN = AS

value is a literal value, or one of the two system strings *TODAY* or *NOW*. See *SYSTEM 2000 QUEST Language*.

record is the name or component number of the target record.

valuestream is a series of item numbers, each followed by its associated value to be added. The system separator must follow each item number and each value. The item numbers need not be in DESCRIBE output sequence. All items must belong to the target record. The default system separator is the asterisk.

DATA see Chapter 8, "Repetitive Processing."

Considerations

- An ASSIGN action can be specified only in a non-tree command.
- When format 1 is used, the item specified must belong to the target record.
- When format 2 is used, the record specified must be the target record and the items in the value stream must belong to that record.
- When a record is specified in an ASSIGN action, if any original data are to be retained, those data must be included in the value stream.
- When format 2 is used and a value stream is not specified, all values are removed from all selected data records.
- If a non-tree command including an ASSIGN action is issued for a damaged data base, the command is not executed.
- Only items and records for which you have U-authority can be specified in an ASSIGN action.
- CALC item values cannot be specified in the ASSIGN command if the UNIQUE option is in effect. See *SYSTEM 2000 QUEST Language* for more information.
- The QUEST language ASSIGN command can also be used to replace values in existing data records. See *SYSTEM 2000 QUEST Language* for more information.

Examples

The commands in the following examples refer to the EMPLOYEE data base.

Example 1

The commands in this example use format 1.

```
ASSIGN OFFICE-EXTENSION EQ 410 XT369* WHERE C1 EQ 1120*:

AS C414 EQ PHYS ED* WH C2 EQ SCHOLL*:

AS C16 EQ 78752* WH C1 EQ 1120*:
```

Example 2

For the effects of this command on one selected data record, see *SYSTEM 2000 QUEST Language*.

```
ASSIGN C411 EQ UNIV OF MICHIGAN* WHERE C1 EQ 1005*:
```

Example 3

The commands in this example use format 2.

```
ASSIGN JOB SKILLS EQ 201* RUSSIAN* 202* EXCELLENT* 203* 15* END*
   WHERE C2 EQ BOWMAN*:

AS C200 EQ 201* PASCAL* 202* EXCELLENT* 203* 3* END*
   WHERE C1 EQ 1050*:
```

Example 4

For the effects of this command on one selected data record, see *SYSTEM 2000 QUEST Language*.

```
ASSIGN C410 EQ 411* UNIV OF MICHIGAN* 412* BA* 414* ECONOMICS*
   415* MATHEMATICS* END* WHERE C1 EQ 1005*:
```

REPLACING A VALUE WITH A NULL: REMOVE

The REMOVE action in a non-tree command replaces values with nulls. A REMOVE action

- affects only selected data records
- removes values for one or all items.

As the format indicates, there are two formats for a REMOVE action. The choice is determined by the activity involved: removing values for a single item or removing values for all items in selected data records.

With format 1, the software checks each selected data record. If the item has a value, the value is removed; otherwise, no action takes place.

With format 2, the software checks each selected data record and removes any values in the record.

Notice that a REMOVE action either removes values for one item or removes values for all items in selected data records. To remove values for some but not all the items in selected records, you must issue a separate REMOVE action for each item to be removed.

A REMOVE action can be specified in any non-tree command. Only the format of REMOVE is given here. Refer to **Action-Clauses in Non-Tree Commands: IF-THEN-ELSE** on page 7-5 for the complete format of non-tree commands.

Format

```
(1) REMOVE item

(2) REMOVE record
```

item is the name or component number of an item in the target record.

record is the name or component number of the target record.

Considerations

- A REMOVE action can be specified only in a non-tree command.
- When format 1 is used, the item specified must belong to the target record.
- When format 2 is used, the record specified must be the target record.
- If a non-tree command including a REMOVE action is issued for a damaged data base, the command is not executed.

Abbreviation: REMOVE = RE

- Only an item or a record for which you have U-authority can be specified in a REMOVE action.
- CALC item values cannot be removed. See *SYSTEM 2000 QUEST Language*.
- To remove data records from the data base, use the REMOVE TREE command in either the QUEUE language or in the QUEST language.
- The QUEST language REMOVE command can also be used to remove values in existing data records.

Examples

The commands in the following examples refer to the EMPLOYEE data base.

Example 1

The commands in this example use format 1.

```
REMOVE CURRENT DEDUCTION WHERE ALL OF (C113 EQ 03/01/78*,
  C1 EQ 1120*):

RE C302 WH C1 EQ 1043*:
```

Example 2

For the effects of this command on one selected data record, see *SYSTEM 2000 QUEST Language*.

```
REMOVE C413 WHERE C1 EQ 1005*:
```

Example 3

The commands in this example use format 2.

```
REMOVE ENTRY WHERE C1 EQ 3005*:

RE C130 WH ALL OF (C131 GE 1.9*, C131 LE 12.3*, C1 EQ 3006*):
```

Example 4

For the effects of this command on one selected data record, see *SYSTEM 2000 QUEST Language*.

```
REMOVE C410 WHERE C1 EQ 1005*:
```

Chapter 6

Updating Data Trees

INTRODUCTION

This chapter describes the syntax of the two tree commands that modify the structure of a data base. You need to understand the concepts presented in Chapter 1, "Using the QUEUE Language: Concepts" in order to understand this chapter.

The two tree commands that modify the structure of a data base are listed below. The list includes the sections where the syntax is described.

- The APPEND TREE command adds new records, new subtrees, or new logical entries to a data base. (See **Adding New Data Trees: APPEND TREE** on page 6-2.)

- The REMOVE TREE command removes data trees (including logical entries) from a data base. That is, it removes data records and their descendants. (See **Removing Data Trees: REMOVE TREE** on page 6-6.)

If only values within existing data records are to be modified, use non-tree commands with the actions described in Chapter 7, "Combining Updates and Retrievals."

The data base cycle number is incremented once for each QUEUE/TERMINATE session including one or more update commands that are successfully executed. This number is displayed in the output of the QUEST language DESCRIBE command.

OUTPUT FROM UPDATE COMMANDS

Update commands do not produce output similar to the output resulting from retrieval commands. Only two types of messages occur as a result of an update command:

- An error message when there is an error in the command. The message appears immediately after the command.

- -373- DATA BASE UNALTERED - message. This message appears immediately after the TERMINATE command when all the update commands within the session that are executed cause no change to the data base. For example, if the following command were issued and the logical entry did not have a C100 record, the software would issue message 373.

```
REMOVE TREE C100 WHERE C1 EQ 2105*:
```

  If you desire verification of the results of an update command, you can issue appropriate retrieval commands after completing the update session.

  The data base is marked internally if any update process completes abnormally, as in such cases as fatal system errors, time limit job abends, operator drops, system drops, and so on. (Syntax errors do not cause damage to a data base.) Subsequently, if you try to perform an update on a data base damaged from a previous job, the following message is issued:

```
-810- WARNING- DATA BASE DAMAGED.  NO UPDATES PERMITTED -
```

  This message appears on the Message File of every job that uses a data base damaged by a previous job abend. The message is also printed if you attempt to use an update command. The data base can still be saved, restored, and described, and retrievals are allowed.

ADDING NEW DATA TREES: APPEND TREE

The APPEND TREE command adds new data trees to a data base. You can add

- new data records to existing logical entries (single data records or data records with descendants)

- new data trees to existing logical entries

- a logical entry to a data base.

There are two formats for the APPEND TREE command. Use format 1 to add a logical entry to a data base; use format 2 to add trees to existing logical entries.

Using format 1 to add a logical entry to a data base is the only case in which a QUEUE language retrieval or update command does not include a where-clause. The new logical entry is appended to the data base; that is, it becomes the last entry in the data base.

If data records (with or without descendants) are to be added to existing logical entries, use format 2. The new records are always placed in last (rightmost) position under their parents. The logical entries to be modified as well as the parents of the new records are determined by the records qualified by the where-clause and by the selected records. The selected records are those records that are both occurrences of the parent of the target record and are related to the qualified data records. The target record is the record specified immediately after APPEND TREE. For a command to be valid, the target record and the focal record must be related.

Format

```
(1) APPEND TREE |CO                EQ |valuestream END* :
                |level-0-record        |*DATA*

(2) APPEND TREE non-level-0-record EQ |valuestream END* where-clause :
                                      |*DATA*
```

level-0-record is the name ENTRY or a new name for the level-0 record.

valuestream is a series of component numbers and associated item values in the same format as used for the loader stream. (See *SYSTEM 2000 QUEST Language, Release 11.6 under IBM and CMS*.) The record number to the left of the EQ operator is not included in the value stream.

non-level-0-record is the name or component number of a non-level-0 record.

DATA See Chapter 8, "Repetitive Processing."

where-clause See Chapter 2, "Specifying Qualification Criteria: The Where-Clause."

Abbreviation: APPEND TREE = AP

Considerations

- If format 1 is used, a where-clause cannot be included.
- If format 2 is used, a where-clause is required.
- Each value specified in an APPEND TREE command must have the system separator (the default is the asterisk) immediately following it.
- When the value stream is not included, an empty record is added to the data base. Also, if a record number is followed immediately by the system separator and another record number in the value steam, an empty record is inserted in the data base for the former record.
- If the APPEND TREE command is issued for a damaged data base, the command is not executed.
- Only items and records for which you have U-authority can be specified in the action-clause of an APPEND TREE command.
- CALC records cannot be appended if the UNIQUE option is in effect. See *SYSTEM 2000 QUEST Language* for information on CALC and UNIQUE.
- The APPEND TREE command adds new data records to a data base. It cannot be used to add new values to existing data records.
- To add data records in other than in the last position under their parents, use the QUEST language INSERT TREE command.

Examples

The examples in this section show the syntax of APPEND TREE commands, using the EMPLOYEE data base. Other examples of APPEND TREE commands can be found in *SYSTEM 2000 QUEST Language*. Some of those examples show the processing and effects of APPEND TREE commands.

Example 1

The following command adds records 32 and 33 for Data Entry Four. Notice that because the value stream does not include values for items in the C0 record, record 32 contains only null values.

```
APPEND TREE C0 EQ 100*101*PROGRAMMER*102*INFORMATION SYSTEMS*
   103*MYJ*104*PROFESSIONAL*END*:
```

Example 2

The following APPEND TREE command adds a new logical entry with one value (for C1) and one data record for each schema record in the data base. All data records, except for the C0 record, would contain only null values.

```
AP C0 EQ 1*3456*100* 110* 120* 130* 200* 300* 400* 410* 420* END*:
```

Example 3

The following APPEND TREE command adds a C130 record as the last (rightmost) record under C100 records for those employees who worked for the Marketing Department on March 31, 1979.

```
AP C130 EQ 131* 10.0* 132* PARTICIPATED IN 1979 ADVERTISING
   CAMPAIGN* END* WH ALL OF (C105 EQ 03/31/79*, C102 EQ MARKETING*):
```

REMOVING DATA TREES: REMOVE TREE

The REMOVE TREE command removes data trees from a data base. You can use REMOVE TREE to

- remove data trees from existing logical entries, that is, selected data records and their descendants
- remove logical entries from the data base.

The selected data records are those records that are both occurrences of the target record and related to the qualified data records. As with the REMOVE TREE command in the QUEST language, the target record must be related to the focal record.

After each data tree is removed, the remaining trees are relinked to close the gap at the level of the specified record. For example, if C10 is the parent record of three C100 records, and the second C100 record is removed, then the third C100 record becomes the second C100 record under the C10 record. The space in the Data Table occupied by the removed records becomes reusable space, although it is not immediately used if more than one user is accessing the data base in a Multi-User environment.

A data record that is removed from the data base during a QUEUE/TERMINATE session cannot be accessed by an APPEND TREE or non-tree commands in the same session. See **Executing the Commands** on page 1-9 for details.

The QUEUE language REMOVE TREE command and the QUEST language REMOVE TREE command are identical in their effect, differing only in their syntax. For instance, the following QUEST language REMOVE TREE command:

```
REMOVE TREE C100 WHERE C102 EQ MARKETING AND C1 EQ 1250:
```

is equivalent to the QUEUE language command shown next.

```
REMOVE TREE C100 WHERE ALL OF (C102 EQ MARKETING*,C1 EQ 1250*):
```

The difference in these commands appears in the where-clauses. The only other syntax difference between the two REMOVE TREE commands is that trace notation cannot be used in update commands in the QUEUE language. Also, message 342 does not appear in QUEUE.

Records added to the data base with APPEND TREE commands cannot use space freed by REMOVE TREE commands issued in the same QUEUE/TERMINATE session. However, they do occupy reusable space that was available before the session began.

Format

```
REMOVE TREE record where-clause:
```

record is the component name or number of the target record.

where-clause See Chapter 2, "Specifying Qualification Criteria: The Where-Clause."

Considerations

- A where-clause is required in all REMOVE TREE commands.
- If the REMOVE TREE command is issued for a damaged data base, the command is not executed.
- Only records for which you have U-authority can be specified in the action-clause of a REMOVE TREE command.
- To remove values from existing records without removing the records, use the REMOVE action described in **Replacing a Value with a Null: REMOVE** on page 5-13.
- To remove records that cannot be identified with a where-clause, use the QUEST Language REMOVE TREE command described in *SYSTEM 2000 QUEST Language*.

Examples

The examples in this section refer to the EMPLOYEE data base.

Example 1

The following command removes the logical entry for employee 1977.

```
REMOVE TREE C0 WHERE C1 EQ 1977*:
```

Example 2

This command removes all C120 records with values for C121 that are equal to or less than 12/31/78. In other words, it removes from the data base all MONTHLY PAYROLL ACCOUNTING records for 1978 or before.

```
RT C120 WH C121 LE 12/31/1978*:
```

Abbreviations: REMOVE TREE = RT

Chapter 7

Combining Updates and Retrievals

INTRODUCTION

This chapter describes the syntax, considerations, and examples of non-tree commands. You need to understand Chapter 1, "Using the QUEUE Language: Concepts" in order to understand this chapter.

Non-tree commands update and retrieve data in selected data records. Selected data records are occurrences of the target record and are also related to the qualified data records. All non-tree commands consist of an action-clause and a where-clause. The action-clause can consist of just one update or retrieval action or of several actions whose execution can depend on whether a given selected data record satisfies the criteria specified in an optional if-clause.

PRINT is the only retrieval action that can be specified in a non-tree command. Its syntax is given in **Retrievals from Data Records: PRINT** on page 4-1. The syntax of the update actions ADD, CHANGE, ASSIGN, and REMOVE is given in Chapter 5, "Updating Data Records."

One important characteristic of the action-clause of a non-tree command is that all items specified in it must belong to the same record, the target record for the command. This implies, of course, that the only record that can be specified in the action-clause of a non-tree command is the target record.

NON-TREE COMMANDS

The execution of non-tree commands basically involves five steps. The first three steps are executed in parallel with all retrieval and update commands in the same QUEUE/TERMINATE session (except for APPEND TREE) as explained in **Processing Terminology** on page 1-6.

1. The Hierarchical Table addresses of the qualified data records are obtained from evaluation of the where-clause.

2. The target schema record is determined from the action-clause.

3. The Hierarchical Table addresses of the selected data records are obtained from the results of the first step. If the qualified data records are already at the level of the target record, the qualified data records become the selected data records. Otherwise, the qualified data records are replaced by their ancestor (or descendant) records that are also occurrences of the target record.

4. The Hierarchical Table addresses of the selected data records are sorted by address, if not already in that order.

5. The selected data records are then processed one at a time. The action-clause is scanned from left to right.

When an if-clause is present, the if-clause tests the values in the selected data records. If they satisfy the if-clause, then the action(s) listed to the right of THEN are applied to those records. When the third format is used (that is, ELSE is included), those actions to the right of ELSE are applied to the selected data records that do not satisfy the if-clause. Otherwise, records that do not satisfy the if-clause are discarded.

When an if-clause is not present, the actions apply to all selected data records.

Examples of Non-Tree Commands

The following three examples assume that the EMPLOYEE data base consists of only two entries: Data Entry One and Data Entry Two in the foldout at the end of the manual. For purposes of illustration, all the commands specify PRINT actions instead of update actions, although the examples apply to both retrievals and updates.

Example 1

In the following command, the focal record is C120, and the target record is its parent (C110); therefore, the command is valid. The qualified data records arranged, by their address in the Hierarchical Table, are 34, 35, 36, 37, 39, 40, 41, 42, 44, and 45.

The selected data records are those C110 records that are parents of the qualified data records: 3, 14, 38. The values for C113 and C114 are retrieved from those records.

```
PRINT C113, PRINT C114 WHERE C121 EXISTS:


113* 10/01/1978  \
114*     $455.65 /  from record 3
113* 03/01/1978  \
114*     $215.40 /  from record 14
113* 04/02/1979  \
114*      $55.73 /  from record 38
```

Example 2

In this command, the focal record is C100, and the target record is its descendant C120; therefore, the command is valid. The qualified data records are 2, 13, and 16. The selected records are those C120 records that are descendants of the qualified data records: 34, 35, 36, 37, 39, 40, 41, 42, 44, and 45. Each of these records is accessed one at a time. The ones containing values for C123 and C126 that meet both if-conditions are 35, 41, and 44. Values for C123 and C126 are retrieved from those records and printed.

```
IF ALL OF (C123 GT 0.0*, C126 LT 1000.0*) THEN PRINT C123, PRINT C126
   WHERE C102 EQ INFORMATION SYSTEMS*:


123*   12.00     \
126*     $984.60 /  from record 35
123*   28.00     \
126*     $944.35 /  from record 41
123*   12.00     \
126*     $944.35 /  from record 44
```

Example 3

In this command, the target record is C110. Because the focal record (C120) is its descendant, the command is valid.

The qualified data records are 3, 14, 15, 17, and 38. The selected data records are the C120 records that are descendants of the qualified data records: 34, 35, 36, 37, 39, 40, 41, 42, 44, and 45. Each of these records is accessed one at a time. PRINT C120 acts on those records whose values for C122 and C123 meet both if-conditions (35, 41, 44, and 45). PRINT C122 acts on the remaining selected records.

```
IF ALL OF (C122 GT 160.00*, C123 GT 0.0*) THEN PRINT C120
   ELSE PRINT C122 WHERE C112 EQ MONTHLY*:

122*  184.00        -->  from record 34

121* 01/31/1979     \
122*  184.00         \
123*   12.00          \
124*   $1,200.00      /  from record 35
125*     $135.60     /
126*     $984.60    /
122*  160.00        -->  from record 36
122*  160.00        -->  from record 37
122*  176.00        -->  from record 39
122*  176.00        -->  from record 40

121* 04/30/1979     \
122*  168.00         \
123*   28.00          \
124*   $1,400.00      /  from record 41
125*     $282.50     /
126*     $944.35    /
122*  168.00        -->  from record 42

121* 05/31/1979    \
122*  184.00        \
123*   12.00         \
124*   $1,400.00     /  from record 44
125*     $282.50    /
126*     $944.35   /

121* 05/31/1979    \
122*  184.00        \
123*   10.00         \
124*   $1,300.00     /  from record 45
125*     $153.20    /
126*   $1,060.35   /
```

ACTION-CLAUSES IN NON-TREE COMMANDS: IF-THEN-ELSE

An action-clause in a non-tree command specifies retrieval and update actions for selected data records. There are three acceptable formats for non-tree commands. Use the one that is appropriate for your task.

- Use format 1 when one or more actions are to be performed on all selected data records.
- Use format 2 when one or more actions are to be performed on all selected data records that satisfy the criteria in the if-clause.
- Use format 3 when one or more actions are to be performed on the selected data records that satisfy the criteria in the if-clause, and one or more actions are to be performed on the selected data records that do not satisfy the criteria in the if-clause.

Format

(1) *actions where-clause* :

(2) *if-clause* **THEN** *actions where-clause* :

(3) *if-clause* **THEN** *actions* **ELSE** *actions where-clause* :

actions	are one or more of the following actions separated by commas: PRINT, ADD, CHANGE, ASSIGN, or REMOVE.
where-clause	See Chapter 2, "Specifying Qualification Criteria: The Where-Clause."
if-clause	See Chapter 3, "Specifying Criteria in Action-Clauses: The If-Clause."

Considerations

- All non-tree commands require a where-clause.
- All items in the action-clause (if-clause and actions) must belong to the target record.
- A comma cannot be specified before the keywords THEN, ELSE, or WHERE.
- A non-tree command cannot contain a tree action. That is, APPEND TREE, PRINT TREE, or REMOVE TREE cannot be specified in a non-tree command.
- See also considerations for the individual actions.
- If one or more actions in the action-clause of a non-tree command contain syntax errors, the command is not executed.

Examples

The examples in this section refer to the EMPLOYEE data base. Other examples of non-tree commands can be found in **Examples of Non-Tree Commands** on page 7-2.

Example 1

The commands in this example use format 1.

```
PRINT C1, CHANGE C3 EQ JOHN* WHERE C2 EQ BROWN*:
PR C300, AD C301 EQ CHESS*, PR C300 WHERE C1 EQ 1120*:
PR C103, CH C103 EQ MYJ*, PR C100 WH C103 EQ MYK*:
```

Example 2

The commands in this example use format 2.

```
IF C101 EQ PROGRAMER* THEN CHANGE C101 EQ PROGRAMMER*
   WHERE C102 EQ INFORMATION SYSTEMS*:
IF ANY OF (C201 EQ PLI*, C201 EQ PL1*, C201 EQ PL/I*)
   THEN CH C201 EQ PL/I*, PR C200 WH C201 EXISTS:
IF C10 GE 600* THEN PR C1, PR C2, PR C3 WH C1 EXISTS:
IF C126* GE C124* THEN PR C120 WH C1 EXISTS:
```

Example 3

The commands in this example use format 3. The last command verifies the Social Security number of all employees whose employee number is in the Data File (see **Accessing the Data File** on page 8-7.) If the Social Security number is incorrect or does not exist, the number in the Data File is entered.

```
IF ANY OF (C414 EXISTS, C415 EXISTS) THEN PR C410, RE C414,
   RE C415 ELSE PR C410 WH C412 EQ HIGH SCHOOL DIPLOMA*:
IF C126* GE C124* THEN PR C120 ELSE PR C124, PR C126
   WH C121 EQ 05/31/79*:
REPEAT $ IF ANY OF (C6 NE *DATA*, C6 FAILS) THEN PR C6,
   AS C6 EQ *DATA*, PR C6 ELSE PR C1, PR C6 WH C1 EQ *DATA*$*:
```

Sample values for the Data File for this command are:

```
434-21-1300*434-21-1300*1265*
441-04-0121*441-04-0121*1120*
462-01-0234*462-01-0234*1043*
```

UPDATE ACTIONS IN NON-TREE COMMANDS

The ADD, ASSIGN, CHANGE, and REMOVE actions that can be specified in the action-clause of a non-tree command have the same effect on existing data records as the equivalent commands in the QUEST language. Refer to *SYSTEM 2000 QUEST Language* for a brief description of the functions of each action. The differences, as in the case of the REMOVE TREE command, are syntactical. The where-clause is different, and trace notation as well as functions cannot be specified in QUEUE language commands. Also, message -342- is not issued in QUEUE. CALC items cannot be updated in queue mode unless the MULTIPLE option is in effect.

One basic difference between update actions in QUEUE non-tree commands and QUEST language updates is that more than one action can be specified in the action-clause of a non-tree command, as described in the **Multiple Actions** section on this page.

A non-tree command cannot access a data record that has been removed from the data base, nor can it access a data record that has been added to the data base in the same QUEUE/TERMINATE session. See **Executing the Commands** on page 1-9.

As with other QUEUE commands, the target record must be related to the focal record for the command to be valid.

The syntax that applies to update actions is given in Chapter 5, "Updating Data Records."

Security Considerations

For an update action or command to be processed by the software, you must have the proper authority for each component specified in the command. U-authority is required for the item or record specified immediately to the right of an APPEND TREE or REMOVE TREE command, ADD, ASSIGN, CHANGE, or REMOVE action. U-authority is also required for any component that is updated by the command or action. Also, you must have W-authority for any component in the where-clause of the command, and if the command includes an if-clause, W-authority is also required for every item specified in it. See *SYSTEM 2000 QUEST Language* for more details on security considerations.

MULTIPLE ACTIONS

Action-clauses in non-tree commands can specify more than one action (ADD, ASSIGN, CHANGE, PRINT, or REMOVE). The actions operate on selected data records. These are occurrences of the target record that are related to the qualified data records. As indicated in **Processing Terminology** on page 1-6, all actions in a non-tree command must refer to the same schema record, either explicitly by specifying the record name or component number, or implicitly by specifying items within the record. For example, in the EMPLOYEE data base, this next command is valid, because all actions refer to the same schema record, C0.

```
PRINT C0, ADD C2 EQ BROWN*,CHANGE C3 EQ JOHN* WHERE C1 EQ 2096*:
```

However, the next command is not valid, because the actions refer to items in different schema records.

```
PRINT C1, ADD C102 EQ MARKETING*, PRINT C110 WHERE C1 EQ 2096*:
```

Non-tree commands are executed in the order in which they occur in the command stream; however, they cannot refer to records that are removed or added to the data base within the same QUEUE/TERMINATE session by REMOVE TREE or APPEND TREE commands.

Also, as explained in **Processing Retrieval and Update Commands** on page 1-8, actions within the same command are executed in the left-to-right order in which they are specified in the action-clause.

For example, the commands shown below retrieve the same values; however, the first command retrieves the value for C2 before the value for C3 from each selected data record, while the second command retrieves the C3 value before the C2 value.

```
PRINT C2, PRINT C3 WHERE C1 EQ 1001*:

PRINT C3, PRINT C2 WHERE C1 EQ 1001*:
```

As mentioned, a non-tree command can include an if-clause to specify criteria for the actions in the action-clause. The if-clause and forms of non-tree commands including if-clauses are discussed in Chapter 3, "Specifying Criteria in Action-Clauses: The If-Clause."

Chapter 8

Repetitive Processing

INTRODUCTION

This chapter describes the syntax associated with the REPEAT command and the *DATA* system string.

THE *DATA* SYSTEM STRING

The *DATA* system string tells the software to stop reading the Command File and read the Data File. When the software finds *DATA* in a command, it goes to the current Data File to read a value (or values). The software then replaces *DATA* with the value (or values) it obtained from the Data File.

As the software scans a command, it expects to find such things as keywords, component labels, and values. If the software finds *DATA* instead of a value followed by an asterisk, it goes to the current Data File and obtains a value followed by an asterisk. On the other hand, if it finds *DATA* instead of a value stream, it goes to the current Data File and obtains a value stream and END*.

The Data File cannot be the same physical file as the Command File. Therefore, before *DATA* can be used in a command, you must be sure that the file containing the data (the Data File) is different from the file containing the command (the Command File). For more information on user files, see *SYSTEM 2000 CONTROL Language and System-Wide Commands, Release 11.5 under IBM OS and CMS*.

Format

---

systemseparator **DATA** *systemseparator*

---

systemseparator is a single, special character. The default is the asterisk. See *SYSTEM 2000 QUEST Language, Release 11.6 under IBM and CMS* for a list of available system separators.

Using *DATA*

Depending on the context of the command, *DATA* represents a value, a set of values for a data record, or a set of values for a data tree.

When *DATA* is specified in a where-clause condition or in an if-condition, it stands for a value followed by an asterisk. It also stands for a value followed by an asterisk in an update action when an item name (or component number) precedes the EQ operator.

When the EQ operator is preceded by a record name (or component number) in an update action (ADD, ASSIGN, or CHANGE) or in an APPEND TREE command, *DATA* stands for a value stream followed by END* or for any part of a value stream to the right of an asterisk and for END*.

Examples

The following examples show where *DATA* can be specified in QUEUE commands. The commands refer to the EMPLOYEE data base.

Example 1

For the command

```
IF C1 EQ 2101* THEN ADD C3 EQ JOHN* WHERE C2 EQ BROWN*:
```

the variations shown below illustrate several uses of *DATA*.

```
IF C1 EQ *DATA* THEN ADD C3 EQ JOHN* WHERE C2 EQ BROWN*:
```
with 2101* in the Data File.

```
IF C1 EQ 2101* THEN ADD C3 EQ *DATA* WHERE C2 EQ BROWN*:
```
with JOHN* in the Data File.

```
IF C1 EQ 2101* THEN ADD C3 EQ JOHN* WHERE C2 EQ *DATA*:
```
with BROWN* in the Data File.

```
IF C1 EQ *DATA* THEN ADD C3 EQ *DATA* WHERE C2 EQ *DATA*:
```
with 2101* JOHN* BROWN* in the Data File.

Example 2

For the command

```
CH C200 EQ 201*COBOL* 202* GOOD* 203*1* END* WH C1 EQ 1120*:
```

the following commands illustrate using *DATA*:

```
CH C200 EQ *DATA* WH C1 EQ 1120*:
```
with 201*COBOL* 202* GOOD* 203*1* END* in the Data File.

```
CH C200 EQ *DATA* WH C1 EQ *DATA*:
```
with 201*COBOL* 202* GOOD* 203*1*END* 1120* in the Data File.

```
CH C200 EQ 201*COBOL* 202**DATA* WH C1 EQ *DATA*:
```
with GOOD* 203*1*END* 1120* in the Data File.

```
CH C200 EQ 201*COBOL*202* GOOD**DATA* WH C1 EQ *DATA*:
```
with 203*1* END* 1120* in the Data File.

Example 3

For the command

```
AP C0 EQ 1*1799*2*BROWN*200*201*COBOL*202*GOOD*END*:
```

DATA could be used in these ways:

```
AP C0 EQ *DATA*:
```
with 1*1799*2*BROWN*200*201*COBOL*202*GOOD*END* in the Data File.

```
AP C0 EQ 1*1799*2*BROWN* *DATA*:
```
with 200*201*COBOL*202*GOOD* END* in the Data File.

```
AP C0 EQ 1*1799*2*BROWN*200**DATA*:
```
with 201*COBOL*202*GOOD* END* in the Data File.

Example 4

For the command

```
AP C100 EQ 101*SECRETARY*110*111*3.75*112*HOURLY*END* WH C1 EQ 1024*:
```

the commands listed below show some ways *DATA* could be used.

```
AP C100 EQ *DATA* WH C1 EQ 1024*:
```
with 101*SECRETARY*110*111*3.75*112*HOURLY*END* in the Data File.

```
AP C100 EQ *DATA* WH C1 EQ *DATA*:
```
with 101*SECRETARY*110*111*3.75*112*HOURLY*END*1024* in the Data File.

```
AP C100 EQ 101*SECRETARY**DATA* WH C1 EQ *DATA*:
```
with 110*111*3.75*112*HOURLY*END* 1024* in the Data File.

```
AP C100 EQ 101*SECRETARY*110**DATA* WH C1 EQ *DATA*:
```
with 111*3.75*112*HOURLY*END* 1024* in the Data File.

```
AP C100 EQ 101*SECRETARY*110*111* *DATA* WH C1 EQ *DATA*:
```
with 3.75*112*HOURLY*END* 1024* in the Data File.

Considerations

- When *DATA* stands for all or part of a value stream, END* must be included in the Data File at the end of the value stream. In other words, END* is always read from the Data File and not from the command. For example, the following use of *DATA* is not allowed:

```
AD C200 EQ *DATA*END* WH C1 EQ 1120*:
```

- In the same way that END* cannot follow *DATA* in a command, no part of a value stream can follow *DATA* in a command. For example, the following uses of *DATA* are not allowed:

```
AD C200 EQ *DATA* 202*GOOD*END* WH C1 EQ 1120*:
AD C200 EQ 201**DATA*202**DATA*END* WH C1 EQ 1120*:
```

- If the Data File was created by using an UNLOAD command or the TERMINATE/UNLOAD command, you should ensure that the file contains an appropriate value for each occurrence of *DATA*. The UNLOAD command (in the QUEST language) produces output in a format suitable for use with the *DATA* system string. For example, this command produces the output shown below it:

```
UNLOAD C13, C1:

106*1043*224*1120*1265*310*1001*. . . .
```

  Notice that the fifth value (1265) is the C1 value in Data Entry Three (see the foldout at the end of this manual) and not a C13 value. This is because C13 is null in that logical entry and UNLOAD output never identifies nulls. If the output shown above is used with the following command, the value for C1 (1265) would be entered as the value for C13 in the data record with C1 EQ 310.

```
REPEAT /ASSIGN C13 EQ *DATA* WHERE C1 EQ *DATA*:/:
```

  One way to avoid this problem is to issue the UNLOAD command with a where-clause that includes conditions with the EXISTS operator. For example,

```
UNLOAD C13, C1 WHERE C13 EXISTS AND C1 EXISTS:
```

- The same care used in specifying a value stream within a command must be used when the value stream is placed in the Data File.

  1. If a value for an item is not to be supplied, then the item number is not included in the value stream. For example, if you do not supply a value for C3, a value stream such as the one listed below is valid.

```
1*1120*2*BROWN*4*01/01/78*. . .
```

However, the next two streams would be invalid:

```
1*1120*BROWN*3*4*01/01/78*. . .
1*1120*BROWN*3**4*01/01/78*. . .
```

As a matter of fact, the two consecutive asterisks in the last value stream would cause the software to stop reading the Data File.

2. To enter a data record without values, you need only to specify the record number. For example, to enter a C100 record without values, value streams such as the following two would be appropriate:

```
1*1120*2*BROWN*100*110*111*3.50*. . .*END*
1*1120*2*BROWN*100*END*
```

ACCESSING THE DATA FILE

Two user files input commands or data into the software. The software reads commands from the Command File and data from the Data File.

The Command File, by default, is named INPUT and identifies the input device as the source of user commands. You can, of course, build an external file of commands on tape or some other device. See *SYSTEM 2000 QUEST Language* for details.

Most QUEUE language commands require that a value or values be specified for the commands to be executed. For example, values are needed in conditions and in if-conditions using binary operators. Values must also be specified in most update commands. There are times, however, when it becomes necessary or convenient to have the values placed in a file prior to issuing the commands.

For example, assume that STREET ADDRESS values in the logical entries of several employees need to be updated. You can build a file consisting of the new addresses with a corresponding employee number as identification, as shown below.

```
1302 LAZY LANE*  1120*
2311 HANSFORD*   1043*
        .
        .
        .
4106 MAIN ST.*   1265*
```

After the values are verified, you can give the file an appropriate DDname such as MYDATA, and invoke this Data File when the update commands are ready to be issued. The command DATA FILE IS invokes the Data File. This command can be issued either before or during a QUEUE/TERMINATE session. Of course, the file specified in the command (in this case MYDATA) must be known to the job before calling the software.

The Collect File is not a user file and cannot be used directly as a Data File; however, the Collect File can be unloaded, and the unloaded output (Report File) can then be used as a Data File. (See the UNLOAD command in *SYSTEM 2000 QUEST Language*.)

Because the values for the commands are in the Data File, and the commands themselves are in the Command File, an indicator is needed to tell the software to go to the Data File to read the value. This indicator is the *DATA* system string. Both the Command File and the Data file are read sequentially. When the first *DATA* is found in a command in the Command File, the software reads the first corresponding value from the Data File and incorporates it into the command. When the next *DATA* is found, the next value is read from the Data File, and so on.

Depending on the context of the command, *DATA* represents a value, a set of values for a data record, or a data tree. When *DATA* appears in a condition or in an if-condition, it necessarily represents one value. That is, it directs the software to read one value from the Data File. When *DATA* follows an EQ in an APPEND TREE command, it represents a data tree. That is, it directs the software to read a value stream up to and including the entry terminator (default END*) from the Data File.

The value stream is a series of component numbers and associated item values in the same format as used for the loader stream. When *DATA* follows EQ in an update action for an item, it represents one value. For example,

```
ADD C2 EQ *DATA* WHERE. . .
```

When *DATA* follows EQ in an update action for a record, it represents a value stream and the entry terminator. In this case, the value stream specifies item numbers and associated values.

The following examples show the use of the *DATA* system string. Notice that in Example 2, you repeat the same command as many times as there are street addresses to be changed. This can get very cumbersome. For this situation, the software offers the REPEAT command, which is discussed in **Command Stream Repetition: REPEAT** on page 8-11.

Example 1

This example shows the contents of a file called MYDATA and the commands in a QUEUE/TERMINATE session that use the values in the file. Assume file MYDATA contains the following values, which refer to data in the EMPLOYEE data base.

```
//MYDATA DD *
INFORMATION SYSTEMS* MYJ* 1120* 301*CHESS*
302*AUSTIN CHESS CLUB*END* 1120*111*3.50*112*HOURLY*
113*04/02/1979*114*55.73*120*121*04/30/1979*122*80.00*
123*.00*124*280.00*125*00*126*$263.20*END* 1265*
201*FORTRAN*202*GOOD*203*1*END* 1120*
/*
//SYSIN DD *
USER, DEMO:
DATA BASE NAME IS EMPLOYEE:
QUEUE:
DATA FILE IS MYDATA:
IF C102 EQ *DATA* THEN CHANGE C103 EQ *DATA* WHERE C1 EQ *DATA*:
CHANGE C300 EQ *DATA* WHERE C1 EQ *DATA*:
APPEND TREE C110 EQ *DATA* WHERE C1 EQ *DATA*:
APPEND TREE C200 EQ *DATA* WHERE C1 EQ *DATA*:
EXIT:
```

When the commands are processed, each appearance of *DATA* directs the software to read the Data File and substitute the appropriate values in the command. The first *DATA* is replaced by INFORMATION SYSTEMS*, the second *DATA* is replaced by MYJ*, the third one by 1120*. The fourth *DATA* (after C300) is replaced by 301*CHESSS*302*AUSTIN CHESS CLUB*END*, and so on. In summary, the same results could be obtained by using the following commands:

```
QUEUE:
IF C102 EQ INFORMATION SYSTEMS* THEN CHANGE C103 EQ MYJ*
   WHERE C1 EQ 1120*:
CHANGE C300 EQ 301*CHESS*302*AUSTIN CHESS CLUB*END*
   WHERE C1 EQ 1120*:
APPEND TREE C110 EQ 111*3.50*112*HOURLY*113*04/02/1979*
   114*55.73*120*121*04/30/1979*122*80.00*
   123*.00*124*280.00*125.00*126*$263.20*END*
   WHERE C1 EQ 1265*:
APPEND TREE C200 EQ 201*FORTRAN*202*GOOD*203*1*END*
   WHERE C1 EQ 1120*:
TERMINATE:
```

Example 2

This example shows the contents of a file called ADDRCHNG and the commands in a QUEUE/TERMINATE session that use those values.

As in Example 1, the software encounters *DATA*, then reads the corresponding value from the Data File and processes the command. In this case, the commands change the street addresses of the employees whose EMPLOYEE NUMBER (C1) is specified in the file.

```
//ADDRCHNG DD *
1302 LAZY LANE*          1120*
2311 HANSFORD*           1043*
4106 MAIN ST.*           1265*
741 CANYONWOOD LANE*     1071*
      :
      :
6935 CHERRY CREEK RD.*   1024*
/*
//SYSIN DD *
USER, DEMO:
DATA BASE NAME IS EMPLOYEE:
DATA FILE IS ADDRCHNG:
QUEUE:
CHANGE C14 EQ *DATA* WHERE C1 EQ *DATA*:
CHANGE C14 EQ *DATA* WHERE C1 EQ *DATA*:
CHANGE C14 EQ *DATA* WHERE C1 EQ *DATA*:
CHANGE C14 EQ *DATA* WHERE C1 EQ *DATA*:
      :
      :
CHANGE C14 EQ *DATA* WHERE C1 EQ *DATA*:
TERMINATE:
EXIT:
```

COMMAND STREAM REPETITION: REPEAT

The REPEAT command directs the software to execute the same command or series of commands as many times as necessary.

You can specify the number of times the command(s) are to be repeated two ways: with a number at the end of the REPEAT command or by placing two successive asterisks in the Data File when the commands within the REPEAT specification include the *DATA* system string. If a number is not specified in the REPEAT command, the repetition can continue up to a million times.

When you issue a REPEAT command within a QUEUE/TERMINATE session, the software begins to process the command(s) within the markers over and over. The software stops the repetition when the number specified with the TIMES option is reached or, if the *DATA* system string is specified in the command(s), when two consecutive asterisks are found in the Data File.

Format

```
REPEAT marker commandstream marker  [n TIMES]:
```

marker is a single, special character not appearing in the enclosed command stream. See *SYSTEM 2000 QUEST Language* for a list of special characters.

commandstream is one or more QUEUE commands, not exceeding 250 characters. The command stream must not contain the marker, or the software will mistake the marker for the end of the command stream. The command stream must not contain another REPEAT command or the TERMINATE command.

n is an integer larger than 1.

Considerations

- A REPEAT command can only be issued in a QUEUE/TERMINATE session.
- If the TIMES option is not used, the software will process the command stream one million times, unless *DATA* is included in the command stream and two consecutive asterisks are found in the Data File. For example, the following command would add a million entries to a data base.

```
REPEAT $ APPEND TREE CO EQ END*: $:
```

Examples

The following examples show the syntax of the REPEAT command. All commands refer to the EMPLOYEE data base.

Example 1

The following command produces three identical lists of the employee names in the Marketing Department.

```
REPEAT /PRINT C2, PRINT C3 WHERE C102 EQ MARKETING:/ 3 TIMES:
```

Example 2

The command stream in the following command includes one command that uses *DATA*. The command is repeated until two consecutive asterisks are found in the Data File. Notice that the marker used is the dollar sign. The slash (/) cannot be used because the first value for C4 (BIRTHDAY) read from the Data File contains slashes. The first slash in the date would mark the end of the command stream and the command would not be processed.

```
REPEAT $ IF ANY OF (C4 NE *DATA*, C4 FAILS) THEN PR C4,
   AS C4 EQ *DATA* WH C1 EQ *DATA*: $ :
```

Example 3

The command stream in the following REPEAT command includes two commands. The commands are repeated until two consecutive asterisks are found in the Data File. Notice that abbreviations are used in order to keep the number of characters below the 250-character limit.

```
REPEAT +IF ANY OF(C14 NE *DATA*,C14 FAIL) THEN PR C1,PR C14,AS C14 EQ
   *DATA* ELSE PR C1,PR C14,PR C15,PR C16,WH C1 EQ *DATA*: IF ANY OF
   (C15 NE *DATA*,C15 FAIL) THEN PR C1,PR C15,AS C15 EQ *DATA* ELSE PR
   C1,PR C14,PR C15, PR C16, WH C1 EQ *DATA*:+:
```

Example 4

The command in this example retrieves the same data as the command in Example 3; however, it retrieves the values from JOB SKILLS (C200) records before the values for C1 and C2. The output of this command can be used to enter new C200 records into a data base by using a REPEAT command such as

```
REPEAT /APPEND TREE C200 EQ *DATA* WHERE ALL OF (C1 EQ *DATA*,C2
   EQ *DATA*):/:
```

REPETITIVE PROCESSING EXAMPLES

The examples that follow show more uses of the REPEAT command, and all include the *DATA* system string. In Examples 1 through 6, the contents of the corresponding Data File are shown, followed by a QUEUE/TERMINATE session with a REPEAT command. The output from the command is shown last. For purposes of illustration, only retrieval commands are used. Example 7 shows the use of the REPEAT command in conjunction with the UNLOAD command.

Example 1

The number of times the command is to be repeated is specified in the REPEAT command. That number corresponds to the number of values in the Data File.

```
//EXMPL1 DD *
1001*1002*1003*
/*
      :
      :
      :
Q:
---
DATA FILE IS EXMPL1:
---
REPEAT $PRINT C1, PRINT C2 WHERE C1 EQ *DATA*: $ 3 TIMES:


----SEQUENCE NUMBER--  1
----SEQUENCE NUMBER--  2
----SEQUENCE NUMBER--  3


---
T:


 --SEQUENCE NUMBER--1

1*  1001
2* WATERHOUSE

 --SEQUENCE NUMBER--2

1*  1002
2* BOWMAN

 --SEQUENCE NUMBER--3

1*  1003
2* SALAZAR
---
```

Example 2

The number of times the command is to be repeated is specified in the REPEAT command. That number is larger than the number of values in the Data File. The repetition stops when the two consecutive asterisks are found.

```
//EXMPL2 DD *
1004*1005*1006**
/*
      :
      :
      .
Q:
---
DATE FILE IS EXMPL2:
---
REPEAT $ PRINT C1, PRINT C2 WHERE C1 EQ *DATA: $5 TIMES:


 ----SEQUENCE NUMBER --  1
 ----SEQUENCE NUMBER --  2
 ----SEQUENCE NUMBER --  3


---
T:


 --SEQUENCE NUMBER--1

1*   1004
2* KNIGHT

 --SEQUENCE NUMBER--2

1*   1005
2* KNAPF

 --SEQUENCE NUMBER--3

1*   1006
2* GARRETT
---
```

Example 3

The number of times the command is to be repeated is specified in the first REPEAT command but not in the second. The first one causes the software to read the Data File three times; the second command causes the software to read the Data File until two consecutive asterisks are found.

```
//EXMPL3 DD *
1007*1008*1009*1010*1011*1012**
/*
      :
      :
      :
Q:
---
DATE FILE IS EXMPL3:
---
REPEAT $ PRINT C1, PRINT C2 WHERE C1 EQ *DATA*: $3 TIMES:


 ----SEQUENCE NUMBER --  1
 ----SEQUENCE NUMBER --  2
 ----SEQUENCE NUMBER --  3


---
REPEAT $ PRINT C1, PRINT C2 WHERE C1 EQ *DATA*: $:


 ----SEQUENCE NUMBER --  4
 ----SEQUENCE NUMBER --  5
 ----SEQUENCE NUMBER --  6


T:


 --SEQUENCE NUMBER--1

1*   1007
2* BOWMAN

 --SEQUENCE NUMBER--2

1*   1008
2* HERNANDEZ

 --SEQUENCE NUMBER--3

1*   1009
2* JONES

 --SEQUENCE NUMBER--4

1*   1010
2* JANET F.

 --SEQUENCE NUMBER--5
      :
      :
      :
---
```

Example 4

The number of times the command is to be repeated is specified in the first REPEAT command but not in the second. The first command causes the software to read the Data File three times. The command DATA FILE IS OTHER specifies an empty file and rewinds file EXMPL4; therefore, when the second REPEAT is processed, it causes the software to read data from the beginning of file EXMPL4 until it finds the two asterisks.

```
//EXMPL4 DD *
1007*1008*1009*1010*1011*1012**
/*
//OTHER DD *
/*
      :
      :
      :
Q:
---
DATE FILE IS EXMPL4:
---
REPEAT $ PRINT C1, PRINT C2 WHERE C1 EQ *DATA*: $3 TIMES:
 ----SEQUENCE NUMBER --  1
 ----SEQUENCE NUMBER --  2
 ----SEQUENCE NUMBER --  3
---
DATA FILE IS OTHER:
---
DATE FILE IS EXMPL4:
---
REPEAT $ PRINT C1, PRINT C2 WHERE C1 EQ *DATA*: $:

 ----SEQUENCE NUMBER --  4
 ----SEQUENCE NUMBER --  5
 ----SEQUENCE NUMBER --  6
 ----SEQUENCE NUMBER --  7
 ----SEQUENCE NUMBER --  8
 ----SEQUENCE NUMBER --  9


T:


 --SEQUENCE NUMBER--1

1*   1007
2* BOWMAN

 --SEQUENCE NUMBER--2

1*   1008
2* HERNANDEZ

 --SEQUENCE NUMBER--3

1*   1009
2* JONES
 --SEQUENCE NUMBER--4

1*   1007
2* VIRGINA P.
```

Example 4 (continued)

```
 --SEQUENCE NUMBER--5

1*   1008
2* JESSE L.

 --SEQUENCE NUMBER--6

1*   1009
2* MICHAEL Y.

 --SEQUENCE NUMBER--7

1*   1010
2* JANET F.

 --SEQUENCE NUMBER--8

1*   1011
2* GWENDOLYN

 --SEQUENCE NUMBER--9

1*   1012
2* PEDRO
---
```

Example 5

The two REPEAT commands in this example do not include the number of times the command is to be repeated. The first causes the software to read the Data File three times. After the value 1024* is read, the software starts scanning the command

```
PRINT C1, PRINT C2 WHERE C1 EQ *DATA*:
```

for the fourth time and assigns sequence number 4 to that command. However, in attempting to read a value from the Data File, it finds an asterisk instead. This causes it to stop processing that command, and it proceeds to the next REPEAT command. Since sequence number 4 has already been used, the software assigns sequence number 5 to the command within the second REPEAT command.

Example 5 (continued)

```
//EXMPL5 DD *
1015*1017*1024**1031*1043*1049**
/*
      :
      :
---
DATA FILE IS EXMPL5:
---
REPEAT $ PRINT C1, PRINT C2 WHERE C1 EQ *DATA*: $:


 ----SEQUENCE NUMBER--  1
 ----SEQUENCE NUMBER--  2
 ----SEQUENCE NUMBER--  3


REPEAT $ PRINT C1, PRINT C3 WHERE C1 EQ *DATA*: $:


 ----SEQUENCE NUMBER--  5
 ----SEQUENCE NUMBER--  6
 ----SEQUENCE NUMBER--  7


T:


 --SEQUENCE NUMBER--1

1*  1015
2* SCHOLL

 --SEQUENCE NUMBER--2

1*  1017
2* WAGGONNER

 --SEQUENCE NUMBER--3

1*  1024
2* MUELLER

 --SEQUENCE NUMBER--5

1*  1031
3* TAI

 --SEQUENCE NUMBER--6

1*  1043
3* MOLLY I.

 -SEQUENCE NUMBER--7

1*  1049
3* SOPHIA
---
```

Example 6

The REPEAT command in this example is repeated until the software finds two consecutive asterisks. Notice that the second asterisk is specified in a separate line. When there is no output for a command, the sequence number is not shown, as in the case of sequence numbers 3 and 7 in this example.

```
//EXMPL6 DD *
CAUCASIAN*78741*BROOKS*BLACK*78741*QUINTERO*
HISPANIC*78741*AMEER*ASIAN*78741*SMITH*
*
/*
      :
      :
Q:
---
DATA FILE IS EXMPL6:
---
REPEAT $ IF C8 EQ *DATA* THEN PRINT C2 WHERE C16 EQ *DATA*:
---
PRINT C8 WHERE C2 EQ *DATA*: $:
---


 ----SEQUENCE NUMBER--  1
 ----SEQUENCE NUMBER--  2
 ----SEQUENCE NUMBER--  3
 ----SEQUENCE NUMBER--  4
 ----SEQUENCE NUMBER--  5
 ----SEQUENCE NUMBER--  6
 ----SEQUENCE NUMBER--  7
 ----SEQUENCE NUMBER--  8


T:


 --SEQUENCE NUMBER--1

2* BOWMAN
2* SCHOLL

 --SEQUENCE NUMBER--2

8* BLACK

 --SEQUENCE NUMBER--4

8* HISPANIC

 --SEQUENCE NUMBER--5

2* QUINTERO

 --SEQUENCE NUMBER--6

8* ASIAN

 --SEQUENCE NUMBER--8

8* BLACK
8* CAUCASIAN
8* CAUCASIAN
---
```

Example 7

This example uses the UNLOAD command to retrieve data in a format suitable for use in a Data File. (The same format can be obtained with the TERMINATE/UNLOAD command mentioned in **Ending a Session: TERMINATE** on page 9-3.)

The first UNLOAD command retrieves values for C1 (EMPLOYEE NUMBER) ordered by C2 (LAST NAME). Because the command is issued when the Report File and the Message File are both the OUTPUT file, the data are displayed after the command.

The next command specifies file RESULTS as the new Report File. When the UNLOAD command is issued again, the data are written to file RESULTS. Because the Data File and the Report File cannot be the same physical file, the next command specifies OUTPUT as the new Report File. This also causes the RESULTS file to be rewound. The DATA FILE IS command specifies RESULTS as the Data File, thereby setting the stage for the data that were retrieved by the UNLOAD command to be used in a QUEUE/TERMINATE session.

The *DATA* within the REPEAT command now refers to the values in file RESULTS. Notice that UNLOAD automatically places an extra asterisk at the end of the output. Thus, you do not have to specify how many times the command is to be repeated, because the repetition will stop when there are no more data available in the Data File.

Example 7 (continued)

```
UNLOAD C1, ORDERED BY C2 WHERE C8 EQ HISPANIC:
---
1049* 1247* 1008* 1112* 1145* 1012* 1217* 1003**
UNLOAD C1, ORDERED BY C2 WHERE C8 EQ HISPANIC:
---
REPORT FILE IS OUTPUT:
---
DATA FILE IS RESULTS:
---
      :
      :
Q:
---
REPEAT $ PRINT C2, PRINT C3 WHERE C1 EQ *DATA*: $:


 ----SEQUENCE NUMBER--  1
 ----SEQUENCE NUMBER--  2
 ----SEQUENCE NUMBER--  3
 ----SEQUENCE NUMBER--  4
 ----SEQUENCE NUMBER--  5
 ----SEQUENCE NUMBER--  6
 ----SEQUENCE NUMBER--  7
 ----SEQUENCE NUMBER--  8


---
T:


 --SEQUENCE NUMBER--  1

 2* FERNANDEZ
 3* SOPHIA

 --SEQUENCE NUMBER--  2

 2* GARCIA
 3* FRANCISCO

 --SEQUENCE NUMBER--  3

 2* HERNANDEZ
 3* JESSE L.

 --SEQUENCE NUMBER--  4

 2* JONES
 3* RITA M.

 --SEQUENCE NUMBER--  5

 2* JUAREZ
 3* ARMANDO

 --SEQUENCE NUMBER--  6
      :
      :
      :


---
```

Chapter 9

Controlling Command Processing

INTRODUCTION

The QUEUE language includes several commands that allow you to control the processing of retrieval and update commands. This chapter describes the syntax of these commands.

- The TERMINATE command marks the end of a set of QUEUE commands.
- The CANCEL QUEUE command cancels the processing of QUEUE commands and returns the QUEST processor to you.
- The STOP commands allow you to specify an upper limit of errors in a QUEUE/TERMINATE session.
- The CONTINUE commands countermand the STOP commands.
- The ENABLE EXECUTION command allows you to have the software scan retrieval and update commands so that only syntactically correct commands are executed.
- The DISABLE EXECUTION command prevents the execution of all update and retrieval commands, although it does allow the software to check for syntax errors.

ENDING A SESSION: TERMINATE

The TERMINATE command marks the end of a QUEUE language session. When the TERMINATE command is issued, the software stops the process of scanning the commands (for syntax errors) in that session and begins to execute those commands. (The execution cannot occur if you have issued a DISABLE EXECUTION command or a STOP AFTER SCAN command.) When execution is completed, the QUEST processor is returned to you. In other words, the TERMINATE command can be followed by any QUEST language command or by any system-wide command.

There are two acceptable formats for the TERMINATE command. The first produces the standard PRINT output for all retrieval commands in the session. That is, values are listed one to a line regardless of whether /PRINT is specified. The second format produces value stream format for all retrieval commands in the session. This type of output is identical to the output produced by the QUEST language UNLOAD command. When /UNLOAD is specified, command sequence numbers are not written on the Report File.

If you want the output from a QUEUE/TERMINATE session on a file other than OUTPUT, the REPORT FILE IS command should be issued prior to the TERMINATE command to divert the output to an alternate Report File. Output from a session always goes to the last Report File specified within the session.

Format

```
(1) TERMINATE  [/PRINT]  :

(2) TERMINATE  [/UNLOAD] :
```

Considerations

- The TERMINATE command can be issued only in QUEUE language.
- The TERMINATE command cannot be issued within a REPEAT command.
- To protect against damage, the software clears the data base pages after all commands in a QUEUE/TERMINATE session are executed. See *SYSTEM 2000 QUEST Language, Release 11.6 under IBM and CMS* for details.

Abbreviations:

PRINT = PR
TERMINATE = T
UNLOAD = UN

Examples

These examples use the various ways of issuing the TERMINATE command.

Example 1

The commands in this example use format 1. The space before PRINT and the space after TERMINATE are optional.

```
TERMINATE:
TERMINATE / PRINT:
T:
T/ PRINT:
TERMINATE /PR:
T / PR:
```

Example 2

The commands in this example use format 2. The space before UNLOAD and the space after TERMINATE are optional.

```
TERMINATE / UNLOAD:
T / UNLOAD:
TERMINATE / UN:
T / UN:
```

CANCELLING A SESSION: CANCEL QUEUE

The CANCEL QUEUE command allows you to obtain the QUEST processor by cancelling the processing of QUEUE commands.

When the CANCEL QUEUE command is issued, the software cancels the processing of QUEUE commands and returns the QUEST processor to you. That is, the CANCEL QUEUE command can be followed by any QUEST language command or by any system-wide command.

Issuing the CANCEL QUEUE command is equivalent to issuing the TERMINATE command to end a QUEUE/TERMINATE session where execution of all commands in the session has been disabled. (See **Controlling Command Execution: ENABLE/DISABLE EXECUTION** on page 9-13.) For example, the following two sets of commands would return the QUEST processor to you after the software has scanned the commands for syntax errors without executing them:

```
QUEUE:                      QUEUE:
DISABLE EXECUTION:          command:
command:                    command:
command:                    command:
command:                    CANCEL QUEUE:
TERMINATE:
```

The CANCEL QUEUE command is especially useful in interactive jobs when you want to stop a QUEUE session without terminating the software session (that is, issuing the EXIT command) and without executing any of the QUEUE language commands. Execution of the commands does not occur even if execution is enabled.

Format

```
CANCEL QUEUE:
```

Abbreviation: QUEUE = Q

Considerations

- The CANCEL QUEUE command can be issued at any time between the QUEUE command and the TERMINATE command. The software issues a non-fatal error message if CANCEL QUEUE is issued after the TERMINATE command.

- In a Multi-User environment, the CANCEL QUEUE command does not change the HOLD or LOCK status of the data base. Since where-clause processing and updates are not done until the TERMINATE command is given, the software puts no hold on the data base while scanning the commands. Also, since the Rollback Log is not in effect until the TERMINATE command is issued, cancelling the session does not affect the Rollback Log. See *SYSTEM 2000 CONTROL Language and System-Wide Commands, Release 11.5 under IBM OS and CMS* for details.

Example

The two acceptable forms of the CANCEL QUEUE command are as follows:

```
CANCEL QUEUE:
```

and

```
CANCEL Q:
```

CONTROLLING EXECUTION IN EVENT OF ERRORS: STOP/CONTINUE

The STOP commands allow you to specify an upper limit of errors in a software session. A CONTINUE command countermands a STOP command within a software session.

There are two forms of the STOP command; each has a corresponding CONTINUE countermand. In this section, STOP IF refers to the command in the first format (CONTINUE IF refers to its countermand), and STOP AFTER SCAN IF refers to the command in the second format (CONTINUE AFTER SCAN IF refers to its countermand).

When you issue a STOP IF command, an error counter is set to zero. This counter is incremented once for each retrieval or update command in error (or once for each update command only, if so specified). If the maximum error count specified in the command is reached, the error message for the last rejected command is followed by an exit from the software session.

To reset a STOP IF command, one of the following must occur:

- A new STOP IF command. The error counter is set to zero even if the first STOP IF command specified UPDATE.
- A CONTINUE IF command. This sets both the error counter and the maximum error count to zero.
- An exit from the session. When a new the session begins, both the error counter and the maximum error count are set to zero.

When you issue a STOP AFTER SCAN command, an error counter (different from the one used for STOP IF commands) is set to zero. This counter is incremented once for each retrieval or update command in error (or once for each update command, if so specified). If the maximum error count (also different from the number set by STOP IF commands) has been reached when you issue the TERMINATE command, no commands are executed, the following message is displayed, and you are given the QUEST processor.

```
-226- ERROR LIMIT EXCEEDED -
```

All commands in the QUEUE/TERMINATE session are examined for syntax errors even after the maximum error count is reached.

To reset a STOP AFTER SCAN command, one of the following must occur:

- A new STOP AFTER SCAN command. The error counter is set to zero even if the first STOP AFTER SCAN specified UPDATE.
- A CONTINUE AFTER SCAN command. This sets both the error counter and the maximum error count to zero.
- An exit from the session. When a new software session begins, both the error counter and the maximum error count are set to zero.

If either a STOP IF command or a STOP AFTER SCAN command has been issued, the error counter and the maximum error count for each command are not altered when you change processors.

Only errors in tree and non-tree commands are counted. The appropriate counter (that is, the counter for the STOP IF commands or the counter for the STOP AFTER SCAN commands) is incremented by one even if two or more errors occur in a command.

In the case of the REPEAT command, the software repeatedly scans the commands within it for syntax errors. Consequently, each time a command is found in error, the appropriate counter is incremented by one.

When UPDATE is specified, only errors in update commands or actions are counted. For an error to be considered an update error, the software must recognize the verb as an update verb. In other words, misspellings of update verbs are not considered update errors. For example, the misspelling of ASSIGN in the following command would not cause the counter to be incremented if UPDATE is specified. The counter would be incremented if UPDATE is not specified.

```
ASIGN C103 EQ MYL* WH C1 EQ 1765*:
```

Also, in the case of non-tree commands, when any part of the command to the left of an update action is in error, the counter is incremented if UPDATE is not specified. If UPDATE is specified, the counter is not incremented even if the update action had errors. For example, the date in the update action of the following command is in error, but the counter is not incremented even if UPDATE is specified. That is because PRINT is misspelled.

```
PRUNT C2, CHANGE BIRTHDAY EQ 03/32/48* WHERE C1 EQ 1765*:
```

Format

```
(1) STOP IF   |ANY UPDATE |COMMAND IS   REJECTED :
              |n          |COMMANDS ARE

    CONTINUE IF  |COMMAND IS    REJECTED  :
                 |COMMANDS ARE

(2) STOP AFTER SCAN IF |ANY UPDATE  |COMMAND IS    REJECTED :
                       |n           |COMMANDS ARE

    CONTINUE AFTER SCAN IF  |COMMAND IS    REJECTED :
                            |COMMANDS ARE
```

n is any integer greater than zero, specifying the maximum error count.

ANY is the default maximum error count; that is, the following three commands are equivalent:

```
STOP IF ANY COMMAND IS REJECTED:
STOP IF 1 COMMAND IS REJECTED:
STOP IF COMMANDS ARE REJECTED:
```

The following three commands are also equivalent:

```
STOP AFTER SCAN IF COMMAND IS REJECTED:
STOP AFTER SCAN IF ANY COMMAND IS REJECTED:
STOP AFTER SCAN IF 1 COMMAND IS REJECTED:
```

Considerations

- STOP IF remains in effect even when you change processors. Therefore, if you issue a STOP IF command in QUEST, it can affect subsequent QUEUE processing. Likewise, if STOP IF is issued in QUEUE, it can affect subsequent QUEST processing.
- STOP AFTER SCAN also remains in effect when you change processors, so it can affect subsequent QUEUE or DEFINE processing.
- The STOP commands remain in effect if you enter the CONTROL processor; however, errors in CONTROL language do not alter the error counter.
- For the use of the STOP/CONTINUE commands in QUEST, see *SYSTEM 2000 QUEST Language*.
- For the use of the STOP/CONTINUE commands in the DEFINE language, see *SYSTEM 2000 DEFINE Language, Release 11.5 under IBM OS and CMS*.

Examples

The following examples show the interaction of STOP commands and CONTINUE commands. For purposes of clarification, the number of errors in the counter and the maximum error count are shown.

Example 1

```
                                        Number[1]    Maximum[1]
                                        in Error     Error
                                        Counter      Count

QUEUE:
STOP IF 2 COMMANDS ARE REJECTED             0           2
   tree or non-tree command with error(s)   1           2
STOP IF 3 COMMANDS ARE REJECTED             0           3
   tree or non-tree command with error(s)   1           3
   tree or non-tree command with error(s)   2           3
   tree or non-tree command with error(s)   3           3
         -error message(s) -
   01/05/78  16:42:57  END SYSTEM 2000 - RELEASE 11.5
```

Example 2

```
                                        Number[2]    Maximum[2]
                                        in Error     Error
                                        Counter      Count
QUEUE:
STOP AFTER SCAN IF 2 COMMANDS ARE REJECTED:  0           2
   tree or non-tree command with error(s)    1           2
STOP AFTER SCAN IF 3 COMMANDS ARE REJECTED:  0           3
tree or non-tree command with error(s)       1           3
error-free tree or non-tree command          1           3
tree or non-tree command with error(s)       2           3
tree or non-tree command with error(s)       3           3
error-free tree or non-tree command          3           3
TERMINATE:
   -226- ERROR LIMIT EXCEEDED -
```

---

[1] Error counter and maximum error count for STOP IF commands.

[2] Error counter and maximum error count for STOP AFTER SCAN commands.

Example 3

```
                                                Number[3]     Maximum[3]
                                                in Error      Error
                                                Counter       Count

QUEUE:
STOP IF 3 COMMANDS ARE REJECTED:                    0             3
   tree or non-tree command with error(s)           1             3
CONTINUE AFTER SCAN IF COMMANDS ARE REJECTED:       1             3
   tree or non-tree command with error(s)           2             3
CONTINUE IF COMMANDS ARE REJECTED:                  0             0
   tree or non-tree commands with error(s)          0             0
STOP IF 2 UPDATE COMMANDS ARE REJECTED:             0             2
   APPEND TREE command with error(s)                1             2
   PRINT TREE command with error(s)                 1             2
   REMOVE TREE command with error(s)                2             2
   -error message(s) -
   01/05/78 16:42:57  END SYSTEM 2000 - RELEASE 11.5
```

Example 4

```
                                                Number[4]     Maximum[4]
                                                in Error      Error
                                                Counter       Count

QUEUE:
STOP AFTER SCAN IF 3 COMMANDS ARE REJECTED:         0             3
   tree or non-tree command with error(s)           1             3
CONTINUE IF COMMANDS ARE REJECTED:                  1             3
   tree or non-tree command with error(s)           2             3
CONTINUE AFTER SCAN IF COMMANDS ARE REJECTED:       0             0
   tree or non-tree command with error(s)           0             0
STOP AFTER SCAN IF ANY UPDATE COMMAND IS REJECTED:  0             1
   PRINT TREE command with error(s)                 0             1
   REMOVE TREE command with error(s)                1             1
   error-free tree or non-tree command              1             1
TERMINATE:
   -226- ERROR LIMIT EXCEEDED -
```

[3] Error counter and maximum error count for STOP IF commands.

[4] Error counter and maximum error count for STOP AFTER SCAN commands.

Example 5

This example shows the functioning of the error counters when you change processors, as well as some problems that can occur if you do not reset them after their use within a processor.

	Number in Error Counter for S.I.[5]	Maximum Error Count for S.I.	Number in Error Counter for S.A.S.[6]	Maximum Error Count for S.A.S
DATA BASE NAME IS TESTDB:	0	0	0	0
STOP IF 5 COMMANDS ARE REJECTED:	0	5	0	0
QUEST language command with error(s)	1	5	0	0
QUEUE:				
tree or non-tree command with error(s)	2	5	0	0
STOP AFTER SCAN IF 4 COMMANDS ARE REJECTED:	2	5	0	4
tree or non-tree command with error(s)	3	5	1	4
error-free tree or non-tree command	3	5	1	4
TERMINATE:				
(error-free command is executed)				
QUEST language command with error(s)	4	5	1	4
STOP IF 3 UPDATE COMMANDS ARE REJECTED:	0	3	1	4
QUEST language update command with error(s)	1	3	1	4
QUEUE:				
PRINT TREE command with error(s)	1	3	2	4
APPEND TREE command with error(s)	2	3	3	4
CONTINUE IF COMMANDS ARE REJECTED:	0	0	3	4
tree or non-tree command with error(s)	0	0	4	4
error-free tree or non-tree command	0	0	4	4
TERMINATE:				
-226- ERROR LIMIT EXCEEDED -				
(error-free command is not executed)				
QUEST language commands	0	0	4	4
QUEUE:				
error-free tree or non-tree command	0	0	4	4
error-free tree or non-tree command	0	0	4	4
TERMINATE:				
-226- ERROR LIMIT EXCEEDED -				
(error-free commands are not executed)				

[5] S.I. = STOP IF

[6] S.A.S. = STOP AFTER SCAN

CONTROLLING COMMAND EXECUTION: ENABLE/DISABLE EXECUTION

The ENABLE/DISABLE commands allow for the scanning of tree and non-tree commands while either enabling or disabling their execution.

After the software scans a tree or non-tree command for syntax errors, it checks an edit flag. If the flag is off (DISABLE), the command is discarded. If the flag is on (ENABLE), the command is executed when the TERMINATE command is issued. That is, when the TERMINATE command is issued, only those commands scanned while ENABLE was in effect are executed.

The ENABLE EXECUTION command sets the flag on. The flag remains on until the DISABLE EXECUTION command is issued even if you change processors. The DISABLE EXECUTION command turns the flag off. It remains off until the ENABLE EXECUTION command is issued even if you change processors.

The flag is set to on (ENABLE) at the beginning of a software session.

Format

```
ENABLE EXECUTION :

DISABLE EXECUTION:
```

Considerations

- When you change processors, the status of the edit flag does not change. Also, the ENABLE/DISABLE EXECUTION commands can be issued in the QUEST language and in the DEFINE language. Therefore, if you issue the QUEUE command when the flag is disabled, subsequent QUEUE commands would not be executed unless you issue the ENABLE EXECUTION command immediately after the QUEUE command. Likewise, if you issue the DEFINE language MAP command when the flag is disabled, both the MAP command and subsequent QUEST language commands would not be executed.

- The internal counters set by STOP IF and CONTINUE IF commands are not affected by the ENABLE/DISABLE EXECUTION commands.

Chapter 10

Invoking Strings

INTRODUCTION

One of the conveniences provided by the software is the capability of storing and invoking strings. Strings allow you to store sequences of the command syntax (part of a command, a complete command, or several commands). A string is then invoked by its component name or number, which the software replaces with the stored syntax of the string invoked.

Strings can be simple or parametric. Simple strings have no parameters and are invoked by their component name or number, enclosed by system separators. Parametric strings contain parameters for which you must supply values when the string or function is invoked.

INVOKING SIMPLE STRINGS

A simple string can store part of a command, a complete command, or even several commands. A simple string can even call one or more other strings. Since the software replaces the call for the string with the string's stored syntax, you must supply the rest of the command syntax when invoking a string that is a partial command.

Example 1 shows how to invoke a string that is a single complete command, and Example 2 shows how to invoke a string that is two complete commands. If the simple string invokes one or more other simple strings, the software does this when the calling string is invoked. Example 4 shows the invoking of a string that calls another string.

Format

```
* component *
```

component is the component name or number (including the C) of the stored strings.

Considerations

- A simple string can be invoked at the beginning of a command or following a keyword or symbol recognized by the software. However, in QUEUE commands, a string cannot be invoked following a system separator (the default is the asterisk) except in the cases shown in the last consideration.

- If a syntax error is encountered in a Self-Contained Facility command, the software inhibits string expansion. Therefore, any command delimiter (usually the colon) coded within a string will not be detected. Including a command delimiter at the end of each command invoking a string ensures that the software will process each command.

- A partial REPEAT command cannot be included in a string. Only complete REPEAT commands can be stored in a string. Also, if a string is invoked within a REPEAT command, it can be invoked only within the markers.

- Strings invoked in a QUEUE/TERMINATE session must store commands or parts of commands that conform to QUEUE syntax. However, a string can store the TERMINATE command, optionally followed by QUEST language commands. Or a string can store an entire QUEUE/TERMINATE session, in which case, it must be invoked in QUEST.

- For the commands listed below, the arrows indicate where a string can be invoked.

```
APPEND TREE C110 EQ 111*1400.00*112*MONTHLY*120*121*05/31/79*
↑           ↑      ↑             ↑           ↑   ↑
   122*184.00*END* WHERE C1 EQ 1560*:
       ↑                 ↑     ↑
REMOVE TREE C300 WHERE C1 EQ 1560*:
↑      ↑   ↑          ↑     ↑
PRINT TREE C100 WH ALL OF (C1 EQ 1560*, ANY OF (C4 GE 01/01/72*,
↑     ↑   ↑       ↑       ↑     ↑     ↑   ↑   ↑     ↑
   C4 LE 12/31/72*)):
   ↑     ↑
IF ANY 1 OF (C11 EQ 80.0*, C11 EQ 90.0*) THEN PRINT C1,
↑  ↑  ↑     ↑       ↑       ↑       ↑           ↑     ↑
   ASSIGN C11 EQ 85.0* ELSE REMOVE C11, ADD C11 EQ 0.0*
   ↑      ↑     ↑           ↑       ↑    ↑   ↑      ↑
   WHERE C1 EQ 1560*:
         ↑     ↑
```

Examples

The examples in this section show how simple strings are invoked. The data base used is the EMPLOYEE data base. For each example, the definitions of the strings invoked precede the command invoking the string.

Example 1

This example invokes a string that is a complete command. Since the command terminator (the colon) is included as part of the string definition, the colon is not required when the string is invoked, although including it does not affect the results.

```
2001* STRNG1 (STRING $AP C100 EQ 102*CORPORATION*110*112*MONTHLY*
   120*END* WH C1 EQ *DATA*:$):
```

Since the string is a complete command, it is invoked by its component number,

```
*C2001*
```

or by its component name:

```
*STRNG1*
```

Example 2

This example invokes a string that is two complete commands. The first command is the same as the one used in Example 1; the second command includes different data in the value stream. Note that the command terminator is again part of the string definition.

```
2002* STRNG2 (STRING $ AP C100 EQ 102*CORPORATION*110*112*MONTHLY*
   120*END* WH C1 EQ *DATA*: AP C100 EQ 102*MARKETING*110*
   112*MONTHLY*120*END* WH C1 EQ *DATA*:$):
```

The string is invoked by

```
*C2002*
```

or

```
*STRNG2*
```

Example 3

This example invokes a string that is a partial command. In this case, the string stores the value stream used in the command in Example 1. Note that the colon is not part of the string definition, so it must be specified when the string is invoked.

```
2003*STRNG3 (STRING $ 102*CORPORATION*110*112*MONTHLY*120*END*$):
```

The string can be invoked as in the following command.

```
AP C100 EQ *C2003* WH C1 EQ 1067*:
```

Example 4

This example invokes a simple string that invokes another simple string. Note that STRNG3 is the same as in Example 3 and that STRNG4 stores the command in Example 1, with the value stream replaced by STRNG3.

```
2003*STRNG3 (STRING $ 102*CORPORATION*110*112*MONTHLY*120*END* $):
2004*STRNG4 (STRING $ AP C100 EQ *STRNG3* WH C1 EQ *DATA*:$):
```

The strings are invoked by issuing either of the following commands:

```
*C2004*
```

or

```
*STRNG4*
```

INVOKING PARAMETRIC STRINGS

A parametric string, like a simple string, can store part of a command, a complete command, or several complete commands. A parametric string can also call one or more other strings.

The difference is that a parametric string includes one or more parameters for which you supply values at the time the string is invoked.

Parametric strings are invoked by issuing the system separator, the string name or component number, followed by the appropriate values for the parameters enclosed in parentheses. For example,

```
*C3001(C1, JONES):
```

Format

```
* component (parameter) :
```

component is the component name or number (including the C) of the stored strings.

parameter are integers from 1 to 31 appearing in any order. The values you supply when invoking a parametric string must be specified in the numeric order of the parameters, regardless of the order they appear in the string and must be separated by commas. Parameters must be enclosed in parentheses.

Considerations

- Strings invoked in a QUEUE/TERMINATE session must store commands or parts of commands that conform to QUEUE syntax. However, a string can store the TERMINATE command, optionally followed by QUEST language commands. Or a string can store an entire QUEUE/TERMINATE session, in which case, it must be invoked in QUEST.

- Component names or numbers, item values, conditions, or if-conditions are examples of values that parameters can be given. In the following commands, arrows indicate where a parameter can be placed.

```
APPEND TREE C110 EQ 111*1400.00*112*MONTHLY*120*121*05/31/79*
           ↑           ↑           ↑               ↑
   122*184.00*END* WHERE C1 EQ 1560*:
         ↑                 ↑     ↑
REMOVE TREE C300 WHERE C1 EQ 1560*:
           ↑          ↑     ↑
PRINT TREE C100 WH ALL OF (C1 EQ 1560*, ANY OF *C4 GE 01/01/72*,
           ↑              ↑     ↑            ↑     ↑
   C4 LE 12/31/72*)):
   ↑     ↑
IF ANY 1 OF (C11 EQ 80.00*, C11 EQ 90.00*) THEN PRINT C1,
      ↑    ↑      ↑       ↑     ↑                  ↑
   ASSIGN C11 EQ 85.00* ELSE REMOVE C11, ADD C11 EQ 0.0*
          ↑      ↑                ↑       ↑      ↑
   WHERE C1 EQ 1560*:
         ↑     ↑
```

- When a REPEAT command is included in a string, the number that precedes TIMES cannot be a parameter.

Examples

The following examples use the EMPLOYEE data base. See also the examples in **Command Stream Repetition: REPEAT** on page 8-11.

Example 1

This example invokes a parametric string that is a complete command. The string has three parameters: the first parameter requires a value for DEPARTMENT (C102), the second parameter requires a value for PAY SCHEDULE (C112), and the third parameter requires a value for EMPLOYEE NUMBER (C1). Because each parameter is followed by an extra asterisk, an asterisk is not included after each value when the string is invoked.

```
2001* PARASTRGN1 (STRING $ AP C100 EQ 102**1**110*112**2**
   120*END* WH C1 EQ *3**:$):
```

The string can be invoked with either of the following commands. The values specified can, of course, be changed.

```
*C2001 (CORPORATION, MONTHLY, 1560):
```

or

```
*PARASTRGN1 (CORPORATION, MONTHLY, 1560):
```

These are equivalent to the following command:

```
AP C100 EQ 102*CORPORATION*110*112*MONTHLY*120*END* WH C1 EQ 1560*:
```

Example 2

This example invokes a parametric string that is a partial command. In this case, the string stores the value stream used in the command in Example 1. The string has two parameters, numbered out of sequence. The first parameter is numbered 2 and requires a value for C102. The second parameter is numbered 1 and requires a value for C112. When the strings are invoked, the values for the parameters are supplied in numerical order, not in the order that they appear in the string.

```
2002*PARASTRGN2 (STRING $ 102**2**110*112**1**120*END* $):
```

The string can be invoked as in the following command.

```
AP C100 EQ *C2002 (MONTHLY, CORPORATION) WH C1 EQ 1560*:
```

The command is equivalent to the command shown in Example 1.

Example 3

This example invokes a parametric string that is a complete command. The string has three parameters: the first parameter requires a value for DEPARTMENT (C102), the second parameter requires a value for PAY SCHEDULE (C112), and the third parameter requires a value for EMPLOYEE NUMBER (C1). Because each parameter is followed by an extra asterisk, an asterisk is not included after each value when the string is invoked.

```
2001* PARASTRGN1 (STRING $ AP C100 EQ 102**1**110*112**2**
   120*END* WH C1 EQ *3**:$):
```

The string can be invoked with either of the following commands. The values specified can, of course, be changed.

```
*C2001 (CORPORATION, MONTHLY, 1560):
```

or

```
*PARASTRGN1 (CORPORATION, MONTHLY, 1560):
```

This is equivalent to the following command.

```
AP C100 EQ 102*CORPORATION*110*112*MONTHLY*120*END* WH C1 EQ 1560*:
```

Example 4

This example invokes a parametric string that is a partial command. In this case, the string stores the value stream used in the command in Example 1. The string has two parameters, numbered out of sequence. The first parameter is numbered 2 and requires a value for C102. The second parameter is numbered 1 and requires a value for C112. When invoking the string, the values for the parameters are supplied in numerical order, not in the order that they appear in the string.

```
2002*PARASTRGN2 (STRING $ 102**2**110*112**1**120*END* $):
```

The string can be invoked as in the following command.

```
AP C100 EQ *C2002 (MONTHLY, CORPORATION) WH C1 EQ 1560*:
```

The command is equivalent to the command shown in Example 1.

Example 5

This example uses the REPEAT command, the *DATA* system string, and parametric strings to issue repetitive commands. Two strings are used in this example. The first string is a parametric string with four parameters. The first parameter requires a value for C14, the second parameter requires a value for C15, the third parameter requires a value for C16, and the fourth parameter requires a value for C1. The second string is a simple string that stores a REPEAT command and invokes the first string.

```
2004*CH-ADDRESS (STRING $CH C14 EQ *1*, CH C15 EQ *2*, CH C16 EQ *3*
   WH C1 EQ *4*:$):
2005*CH-ADDRESSES (STRING $REPEAT /*C2004 (*DATA*, *DATA*, *DATA*,
   *DATA*):/:$):
```

- String CH-ADDRESS can be used to change one employee's address at a time. For example,

  ```
  *C2004 (2311 MAIN ST.*, AUSTIN TX*, 78742*, 1013*):
  ```

  changes the address for employee 1013. Because the parameters in the string definition do not include an extra asterisk, an asterisk must be provided with each value when the string is invoked. Otherwise, if the string is invoked as follows,

  ```
  *C2004 (2311 MAIN ST., AUSTIN TX, 78742, 1013):
  ```

  the equivalent command is

  ```
  CH C14 EQ 2311 MAIN ST., C15 EQ AUSTIN TX, C16 EQ 78742 WH
     C1 EQ 1013:
  ```

  This command cannot be processed because the values are not followed by asterisks. However, if each argument in the string definition had been followed by an extra asterisk (as was done in the other examples), for example, *1**, then the string could be invoked without specifying asterisks after the values.

- The reason for not placing extra asterisks after the arguments in the definition of string C2004 is that string C2004 is invoked in string C2005, which uses *DATA*. Because *DATA* stands for a value followed by an asterisk, placing an extra asterisk after each argument in the definition of string C2004 would have caused an error when *DATA* was replaced in the command.

  String C2005 can be issued once a file is specified as the current Data File. Assume that file ADDRESS contains the following data.

  ```
  2311 MAIN ST.* AUSTIN TX*78742*1013*
  5601 LONG DR*AUSTIN TX*78754*1016*
  *
  ```

  When the following commands are issued, the software processes the REPEAT command in string C2005, repeating the command stored in string C2004 with the values in file ADDRESS.

  ```
  DATA FILE IS ADDRESS:
  QUEUE:
  *CH-ADDRESSES*
  ```

When *C2005* is issued, the software processes the command stored in the string. That is,

```
REPEAT /*C2004 (*DATA*,*DATA*,*DATA*,*DATA*):/:
```

When it finds *C2004, it places the command stored in string C2004 within the REPEAT command, replacing the arguments with *DATA*.

```
REPEAT /CH C14 EQ *DATA*, CH C15 EQ *DATA*,CH C16 EQ *DATA*
   WH C1 EQ *DATA*:/:
```

Thus, as the software scans the command from left to right, it replaces each *DATA* with the value (and asterisk) obtained from the Data File. When the software finds two consecutive asterisks in the Data File, it stops the repetition.

The two commands processed as a result of invoking string C2005 are the following:

```
CH C14 EQ 2311 MAIN ST.*,CH C15 EQ AUSTIN TX*,CH C16 EQ 78742*
   WH C1 EQ 1013*:
```

and

```
CH C14 EQ 5601 LONG DR*,CH C15 EQ AUSTIN TX*,CH C16 EQ 78754*
   WH C1 EQ 1016*:
```

Chapter 11

Comparing the QUEST and QUEUE Languages

INTRODUCTION

This chapter compares the syntax of the QUEST language with the syntax of the QUEUE language. Examples throughout show the differences between the two languages. The commands in the examples refer to the EMPLOYEE data base shown in the foldout at the end of this manual.

GENERAL DIFFERENCES

This section compares the syntax of the two languages for update and retrieval commands.

	QUEST	QUEUE
1.	Values are followed by asterisks only in the action-clauses of update commands. `AS C2 EQ BROWN* WH C1 EQ 1014:`	Values are always followed by asterisks. `PR C2 WH C1 EQ 1014*:`
2.	LIMIT can be used to limit the number of records affected by retrieval or update commands. `LIMIT 2, 10:`	The LIMIT command is not available.
3.	The system strings *NOW* and *TODAY* can be used as values in QUEST commands. `AD C104 EQ *TODAY* WH C1 EQ 1014:`	The system strings *NOW* and *TODAY* can be used as values in QUEUE commands. `AD C104 EQ *TODAY** WH C1 EQ 1014*:`
4.	The system string *DATA* cannot be used.	The system string *DATA* tells the software to read the value from the current Data File. `AD C104 EQ *DATA* WH C1 EQ *DATA*:`
5.	Strings can be used in QUEST as long as the stored syntax conforms to planned use.	Strings can be used in QUEUE as long as the stored syntax conforms to planned use.

6.	The REPEAT command is not available.	REPEAT directs multiple executions of commands.
		
		`REPEAT /AD C14 EQ *DATA*`
		`WH C1 EQ *DATA*:/:`

ACTION-CLAUSES

This section compares the syntax of the action-clause in update and retrieval commands.

	QUEST	QUEUE

1.	Multiple actions are not allowed in one command. However, one retrieval verb can be used to retrieve values for several components. Also, the SAME operator allows the use of the where-clause in several commands.
	
	`PR C2, C3 WH C1 EQ 1014:`
	`CH C2 EQ BROWN* WH SAME:`
		Multiple non-tree actions are allowed in one command. All components must be in the same record and the verb must be repeated. The SAME operator is not available in QUEUE.
		
		`PR C2, PR C3, CH C2 EQ BROWN*`
		`WH C1 EQ 1014*:`

2.	DITTO allows the use of the same action-clause in several consecutive commands.
	
	`AD C120 EQ 121*05/31/79*122*184.*`
	`END* WH C1 EQ 1001:`
	`DITTO WH C1 EQ 1002:`
	`DITTO WH C1 EQ 1003:`
	. . .
		DITTO is not allowed. However, in some cases, the REPEAT command and the *DATA* system string can be used to repeat the action-clause.
		
		`REPEAT /AD C120 EQ 121*05/31/79*`
		`122*184.*END* WH C1 EQ *DATA*:/:`
		with 1001*1002*1003*...** in the current Data File.

3.	If-clause syntax is not available.	The if-clause allows the specification of criteria for actions on selected data records.
		
		`IF C7 EQ MALE* THEN PR C1`
		`WH C8 EQ BLACK*:`

4. IF-THEN-ELSE format is not available.

   The IF-THEN-ELSE format allows the specification of different actions on selected data records.

```
IF C7 EQ FEMALE THEN PR C2,
   PR C5 ELSE PR C1 WH
   C102 EQ MARKETING*:
```

5. System functions can be used in the action-clauses of retrieval and update commands.

```
PR SUM C126 WH C1 EQ 1014:
```

   System functions are not allowed.

6. Ad hoc functions can be used in the action-clauses of retrieval and update commands.

```
AD C126 = (C124 - C114)
   WH C1 EQ 1014:
```

   Ad hoc functions are not allowed.

7. Stored functions can be used in the action-clauses of retrieval and update commands.

   Stored functions are not allowed.

8. The system string *FTODAY* can be used in the action-clause of retrieval and update commands.

```
PR (*FTODAY* - C105)
   WH C1 EQ 1014:
```

   FTODAY is not allowed.

WHERE-CLAUSES

This section compares the syntax of the where-clause in update and retrieval commands. In general, where-clause syntax in QUEUE is more restrictive and offers fewer options than where-clause syntax in QUEST. However, in non-tree commands, the if-clause can be used to express some criteria that cannot be specified in QUEUE where-clauses but can be specified in QUEST where-clauses. QUEUE where-clause syntax and if-clause syntax are compared in Illus. 3.1 on page 3-5.

	QUEST	QUEUE
1.	A where-clause is optional in retrieval commands. It is required in update commands without a full trace. `PR C1 WH C7 EQ FEMALE:`	A where-clause is required in all commands, except APPEND TREE C0. `PR C1 WH C7 EQ FEMALE*:`
2.	Both key items and non-key items can be specified in a where-clause. `PR C1 WH C3 EQ JOHN AND C2 EQ SMITH:`	Only key items can be specified in a where-clause. However, both key and non-key items can be specified in an if-clause. `IF C3 EQ JOHN* THEN PR C1 WH C2 EQ SMITH*:`
3.	The EXISTS and FAILS unary operators can be used to form conditions in a QUEST language where-clause. `PR C1 WH C4 EXISTS AND C13 FAILS:`	Only the EXISTS operator can be used to form conditions in a QUEUE language where-clause. However, both the EXISTS and FAILS operators can be used in an if-clause. `IF C13 FAILS THEN PR C1 WH C4 EXISTS:`

4. Binary operators can be used in a where-clause to compare a data item with a value or a data item with another data item.

```
PR C1 WH C11* EQ C12* AND
   C4 EQ 02/01/78:
```

Binary operators can be used in a where-clause to compare a data item with a value. However, binary operators in an if-clause can be used to compare data items with values or to compare two data items. This item-to-item comparison is more restrictive than the one available in QUEST language where-clauses.

```
IF C11* EQ C12* THEN PR C1
   WH C4 EQ 02/01/78*:
```

5. Ternary operators can be used in where-clauses to compare a data item with a range of values.

```
PR C1 WH C13 SPANS 120 * 130:
```

Ternary operators are not available. The ALL OF operator and appropriate binary operators must be used to compare a data item with a range of values.

```
PR C1 WH ALL OF (C13 GE 120*,
   C13 LE 130*):
```

6. The CONTAINS operator is used to search for characters within values.

```
PR C1 WH C14 CONTAINS AUSTIN:
```

The CONTAINS operator is not available.

7. The HAS operator is used to raise or lower the level of the qualified records for an expression.

```
PR C101 WH C0 HAS C201 EQ COBOL:

PR C1 WH C0 HAS C201 EQ COBOL
   AND C0 HAS C201 EQ PL/I:

PR C101 WH
   (C100 HAS C1 EQ 1014) AT 1:
```

The HAS operator is used to raise the level of the qualified records for a condition.

```
PR C101 WH C0 HAS C201 EQ COBOL*:

PR C101 WH ALL OF (C0 HAS C201 EQ
   COBOL*, C0 HAS C201 EQ PL/I*):
```

8. The AT operator qualifies data records by their position relative to their similar siblings.

```
PR C1, C120 WH C121 EXISTS AT 5:
```

The AT operator is not available.

9. The NOT operator qualifies data records that do not satisfy an expression.

```
PR C2 WH NOT C13 EQ 225:
```

The NOT operator is not available. However, in some cases, the required results can be obtained by using the IF-THEN-ELSE logic in the action-clause.

```
IF C13 EQ 225* THEN PR C1 ELSE
   PR C2 WH C1 EXISTS:
```

10. The AND and OR logical operators combine expressions in a where-clause.

```
PR C1 WH C7 EQ FEMALE AND
   C8 EQ BLACK:

PR C1 WH C7 EQ FEMALE OR
   C8 EQ BLACK:
```

The AND and the OR operators cannot be used. However, expressions and conditions can be combined using the threshold operators ALL OF or ANY OF.

```
PR C1 WH ALL OF (C7 EQ FEMALE*,
   C8 EQ BLACK*):

PR C1 WH ANY OF (C7 EQ FEMALE*,
   C8 EQ BLACK*):
```

11. The threshold operators ALL OF and ANY OF cannot be used. However, the AND and OR operators combine expressions.

```
PR C1 WH (7 EQ FEMALE AND C8
   EQ BLACK)
OR (C7 EQ FEMALE AND C9 EQ
   FULL TIME)
OR (C8 EQ BLACK AND C9 EQ
   FULL TIME):
```

The threshold operators ALL OF and ANY OF can combine up to seven expressions or up to seven conditions.

```
PR C1 WH ANY 2 OF (C7 EQ FEMALE*,
   C8 EQ BLACK*, C9 EQ FULL TIME*):
```

12. The SAME operator repeats where-clause expressions in several consecutive commands.

```
AD C2 EQ BROWN* WH C1 EQ 1014:
CH C3 EQ JOHN* WH SAME:
REMOVE TREE C130 WH SAME:
```

The SAME operator is not available. However, in some cases several actions can be specified in one non-tree command.

```
AD C2 EQ BROWN*, CH C3 EQ JOHN*
   WH C1 EQ 1014*:
REMOVE TREE C130 WH C1 EQ 1014*:
```

13. The delimiter specifies values containing the software keywords.

```
PR C1 WH C411 EQ /COLLEGE OF
   WILLIAM AND MARY/:
```

The delimiter cannot be used. However, an asterisk always marks the end of a value, and the blank after a binary operator marks the beginning of a value.

```
PR C1 WH C411
   EQ COLLEGE OF WILLIAM AND MARY*:
```

RETRIEVAL COMMANDS

This section compares the syntax of retrieval commands. In general, the syntax of retrieval commands in QUEUE is more restrictive and offers fewer options than the syntax of QUEST language retrieval commands.

	QUEST	QUEUE

1.	DESCRIBE displays one or more components in the data base definition or a Collect File.
	
	`DESCRIBE C1 THRU C100:`
	`DESCRIBE COLLECT FILE:`
		The DESCRIBE command is not available.

2.	TALLY displays values for key items. The output also shows how often a value occurs. `TALLY C1, C102:`	The TALLY command is not available.
3.	PRINT can retrieve data from records on several levels. `PR C2, C101, C121 WH C102 EQ MARKETING:`	PRINT can retrieve data from similar records only. `PR C2, PR C3 WH C102 EQ MARKETING*:`
4.	The PRINT TREE command is not available. However, when the TREE format option is in effect, PRINT retrieves the data from data trees whose top record is specified in the retrieval-clause. `PR C100 WH C1 EQ 1014:`[1]	PRINT TREE retrieves the data from data trees whose top record is specified in the action-clause. `PRINT TREE C100 WH C1 EQ 1014*:`

Note: the TREE format option is the default; therefore, it is not specified in this command. If the RECORD format option were in effect, the command could be issued as

PR /TREE/ C100 WH C1 EQ 1014:

5. LIST displays data in columnar format. `LI C1, C2, C3 WH C7 EQ FEMALE:`	LIST is not available.

6. UNLOAD retrieves data in loader stream format.
 	
`UN C100, C1 WH C7 EQ FEMALE:`
	The UNLOAD command is not available. However, when UNLOAD is specified in a TERMINATE command, the output from retrieval commands in the QUEUE/TERMINATE session appears in loader stream format.
	
	`QUEUE:`
	`PRINT TREE C100 WH C7 EQ FEMALE*:`
	`TERMINATE / UNLOAD:`

7. The ordering-clause can be used to order output in retrieval commands with a where-clause.
 	
`PR C2, C3, ORDERED BY C2, C3`
`WH C7 EQ FEMALE:`
	The ordering-clause is not available.

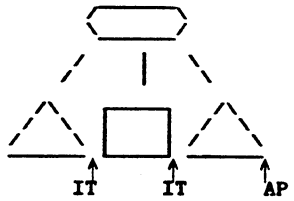
8. Format options can be specified in the FORMOP command or in retrieval commands to control output layout.
 	
`PR /NULL/ C1, C2`
`WH C7 EQ FEMALE:`
	Format options cannot be specified. However, some format options specified in QUEST affect the output of QUEUE commands.
	
	`PR /NULL/:`
	`QUEUE:`
	`PR C1, PR C2 WH C7 EQ FEMALE*:`
	`TERMINATE:`

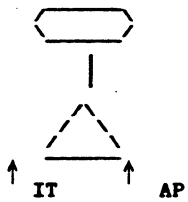
9. The COLLECT command creates a Collect File of values gathered from the data base.	The COLLECT command is not available. Also, the Collect File cannot be accessed with the QUEUE language.

UPDATE COMMANDS

This section compares update commands in the two languages. In general, the main difference is that in QUEST, you can specify the position where an update is to take place. The QUEUE language, however, allows specification of position by content only. Thus, QUEST language update commands can include trace notation in the action-clause, the AT operator in the where-clause, and in the case of INSERT TREE commands, BEFORE and AFTER instead of WHERE. In QUEUE, you must use a where-clause to qualify those records that will be affected by an update command.

The QUEST language INSERT TREE command positions new records differently than the QUEUE language APPEND TREE command. INSERT TREE places new records in the specified position relative to their similar siblings. APPEND TREE places new records in the last position relative to their siblings. This difference is shown in the data trees below. The arrows with IT point to the positions in the data trees where a new 'square' record can be placed with INSERT TREE commands. The arrows with AP point to the positions in the data trees where a new 'square' record can be placed with an APPEND TREE command.





Another difference between the two languages is as follows. In QUEST, the software issues a message indicating the number of selected data records after each syntactically correct command. The effect of the update on the data base can then be determined by issuing appropriate retrieval commands. In contrast, no message is issued in QUEUE language after syntactically correct commands. The effect of an update can be determined only by issuing appropriate retrieval commands.

However, in QUEUE, a non-tree command can include both retrieval and update actions. For example,

```
PR C3, CH C3 EQ JOHN*, PR C3 WH C1 EQ 1014*:
```

	QUEST	QUEUE

1.	Trace notation can be used to specify the position of the record(s) to be updated.
	
	`AD C121 *3 EQ 03/31/79*`
	`  WH C1 EQ 1014:`
		Trace notation is not available.

2. ADD commands are used to give values to nulls. `AD C13 EQ 210* WH C1 EQ 1014:`	ADD actions are used to give values to nulls. `AD C13 EQ 210* WH C1 EQ 1014*:`
3. CHANGE commands are used to replace values. `CH C2 EQ BROWN* WH C2 EQ BROWNE:`	CHANGE actions are used to replace values. `CH C2 EQ BROWN* WH C2 EQ BROWNE*:`
4. ASSIGN commands replace values or nulls with values. `AS C121 EQ 05/31/79* WH C121 LT 05/01/79:`	ASSIGN actions replace values or nulls with values. `AS C121 EQ 05/31/79* WH C121 LT 05/01/79*:`
5. REMOVE commands replace values with nulls. `RE C13 WH C1 EQ 1014:`	REMOVE actions replace values with nulls. `RE C13 WH C1 EQ 1013*:`
6. ASSIGN TREE commands alter the values and/or the structure of existing data trees. `AT C400 EQ 412*MBA*END* WH WH C1 EQ 1014:`	ASSIGN TREE is not available.
7. INSERT TREE commands insert new data records (and their descendants) in specified positions relative to their similar siblings. `IT C0 EQ 1*1014*END* AFTER C1 EQ 1013:`	INSERT TREE is not available.
8. APPEND TREE is not available.	APPEND TREE commands insert new data records (and their descendants) in last position relative to their siblings. `AP C200 EQ 201*COBOL*203*1*END* WH C1 EQ 1014*:`

9. REMOVE TREE commands remove data trees from a data base.

```
RT C130 WH C1 EQ 1014:
```

REMOVE TREE commands remove data trees from a data base.

```
RT C130 WH C1 EQ 1014*:
```

10. MOVE TREE commands move a data tree from one node in the data base to another. The effect is that of performing a REMOVE TREE followed by an INSERT TREE.

```
MOVE TREE C100
   WHERE C1 EQ 1043
   AND C201 EQ GRAPHICS
   TO *0 WHERE C1 EQ 1265:
```

MOVE TREE is not allowed.

11. PREVIOUS can be used to repeat a value or a value stream in several consecutive update commands.

```
CH C120 EQ 121*05/31/79*END*
   WH C1 EQ 1014:
CH C120 EQ PREVIOUS
   WH C1 EQ 1015:
```

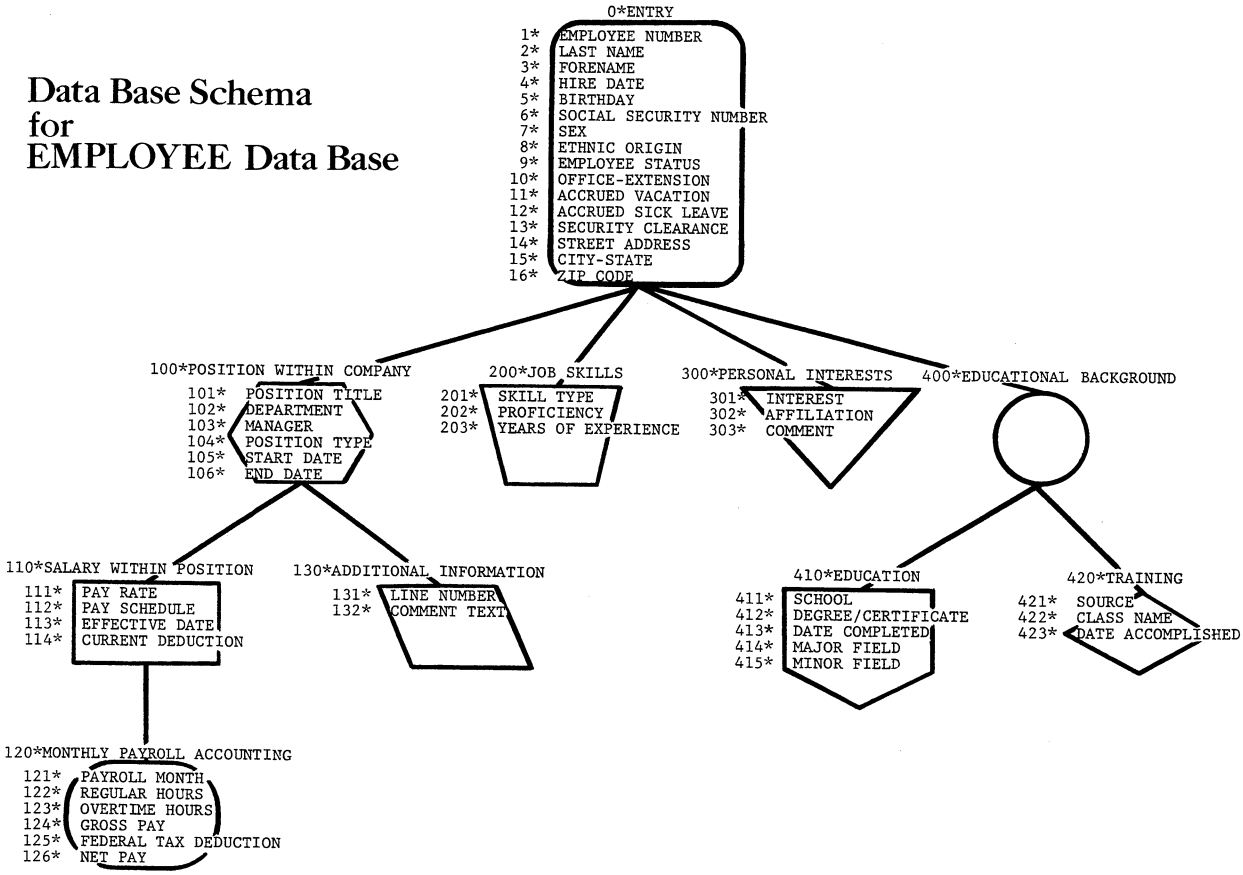
PREVIOUS is not allowed.

Index

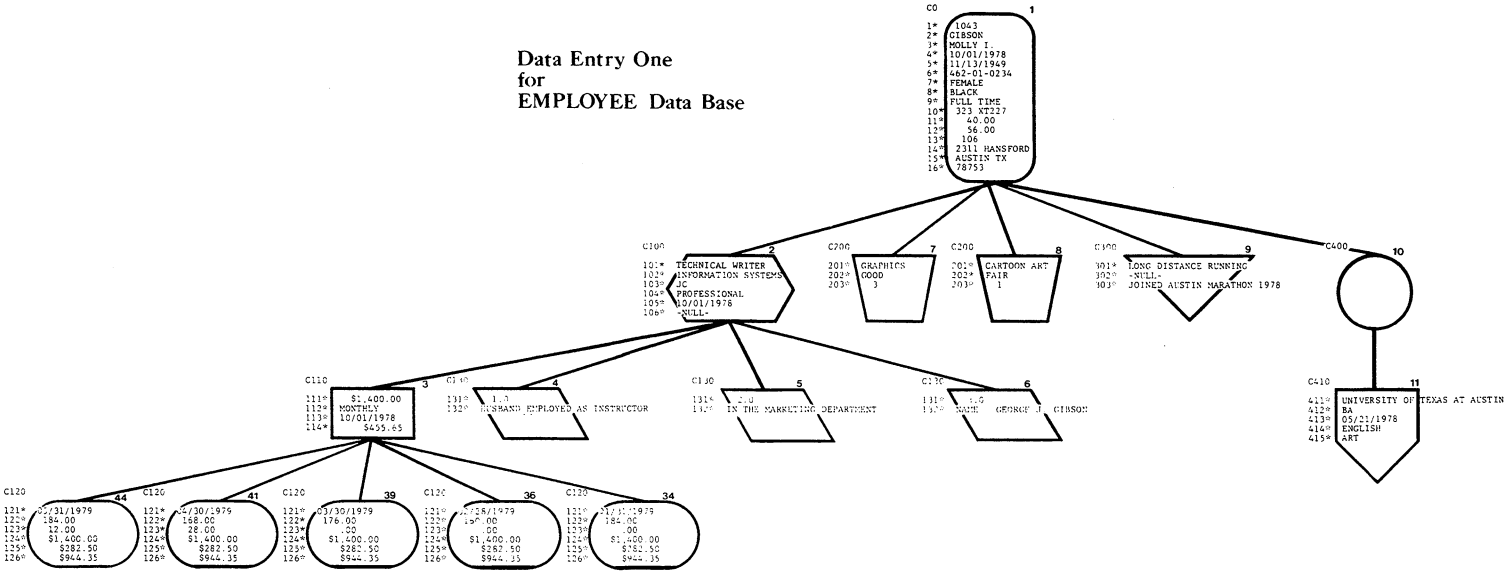
Actual DESCRIBE Output from EMPLOYEE Data Base

```
---
DESCRIBE:
SYSTEM RELEASE NUMBER  XXX
DATA BASE NAME IS      EMPLOYEE
DEFINITION NUMBER             2
DATA BASE CYCLE NUMBER       24
     1*  EMPLOYEE NUMBER (INTEGER NUMBER 9999)
     2*  LAST NAME (CHAR X(10) WITH  FEW FUTURE OCCURRENCES )
     3*  FORENAME (NON-KEY CHAR X(20))
     4*  HIRE DATE (DATE)
     5*  BIRTHDAY (DATE)
     6*  SOCIAL SECURITY NUMBER (NON-KEY CHAR X(11))
     7*  SEX (CHAR X(6) WITH MANY FUTURE OCCURRENCES )
     8*  ETHNIC ORIGIN (CHAR X(9) WITH SOME FUTURE OCCURRENCES )
     9*  EMPLOYEE STATUS (CHAR X(9) WITH MANY FUTURE OCCURRENCES )
    10*  OFFICE-EXTENSION (NON-KEY CHAR X(9))
    11*  ACCRUED VACATION (NON-KEY DECIMAL NUMBER 999.99)
    12*  ACCRUED SICK LEAVE (NON-KEY DECIMAL NUMBER 999.99)
    13*  SECURITY CLEARANCE (INTEGER NUMBER 999 WITH MANY FUTURE OCCURR
         ENCES )
    14*  STREET ADDRESS (NON-KEY CHAR X(20))
    15*  CITY-STATE (NON-KEY CHAR X(15))
    16*  ZIP CODE (CHAR X(5) WITH  FEW FUTURE OCCURRENCES )
   100*  POSITION WITHIN COMPANY (RECORD)
     101*  POSITION TITLE (NON-KEY CHAR X(10) IN 100)
     102*  DEPARTMENT (CHAR X(14) IN 100 WITH SOME FUTURE OCCURRENCES )

     103*  MANAGER (CHAR XXX IN 100 WITH  FEW FUTURE OCCURRENCES )
     104*  POSITION TYPE (CHAR X(12) IN 100 WITH SOME FUTURE OCCURRENCE
           S )
     105*  START DATE (DATE IN 100)
     106*  END DATE (NON-KEY DATE IN 100)
     110*  SALARY WITHIN POSITION (RECORD IN 100)
       111*  PAY RATE (MONEY $9999.99 IN 110)
       112*  PAY SCHEDULE (CHAR X(7) IN 110)
       113*  EFFECTIVE DATE (DATE IN 110)
       114*  CURRENT DEDUCTION (NON-KEY MONEY $9999.99 IN 110)
       120*  MONTHLY PAYROLL ACCOUNTING (RECORD IN 110)
         121*  PAYROLL MONTH (DATE IN 120)
         122*  REGULAR HOURS (NON-KEY DECIMAL NUMBER 999.99 IN 120)
         123*  OVERTIME HOURS (NON-KEY DECIMAL NUMBER 999.99 IN 120)
         124*  GROSS PAY (NON-KEY MONEY $9999.99 IN 120)
         125*  FEDERAL TAX DEDUCTION (NON-KEY MONEY $9999.99 IN 120)
         126*  NET PAY (NON-KEY MONEY $9999.99 IN 120)
     130*  ADDITIONAL INFORMATION (RECORD IN 100)
       131*  LINE NUMBER (DECIMAL NUMBER 99.9 IN 130)
       132*  COMMENT TEXT (NON-KEY TEXT X(7) IN 130)
   200*  JOB SKILLS (RECORD)
     201*  SKILL TYPE (CHAR X(12) IN 200 WITH SOME FUTURE OCCURRENCES )

     202*  PROFICIENCY (NON-KEY CHAR X(5) IN 200)
     203*  YEARS OF EXPERIENCE (NON-KEY INTEGER NUMBER 99 IN 200)
   300*  PERSONAL INTERESTS (RECORD)
     301*  INTEREST (CHAR X(12) IN 300 WITH  FEW FUTURE OCCURRENCES )
     302*  AFFILIATION (NON-KEY CHAR X(5) IN 300)
     303*  COMMENT (NON-KEY TEXT X(5) IN 300)
   400*  EDUCATIONAL BACKGROUND (RECORD)
     410*  EDUCATION (RECORD IN 400)
       411*  SCHOOL (CHAR X(15) IN 410)
       412*  DEGREE/CERTIFICATE (CHAR X(7) IN 410 WITH  FEW FUTURE OCCU
             RRENCES )
       413*  DATE COMPLETED (DATE IN 410)
       414*  MAJOR FIELD (NON-KEY CHAR X(16) IN 410)
       415*  MINOR FIELD (NON-KEY CHAR X(12) IN 410)
     420*  TRAINING (RECORD IN 400)
       421*  SOURCE (NON-KEY CHAR X(12) IN 420)
       422*  CLASS NAME (CHAR X(12) IN 420 WITH  FEW FUTURE OCCURRENCES
             )
       423*  DATE ACCOMPLISHED (DATE IN 420)
---
```

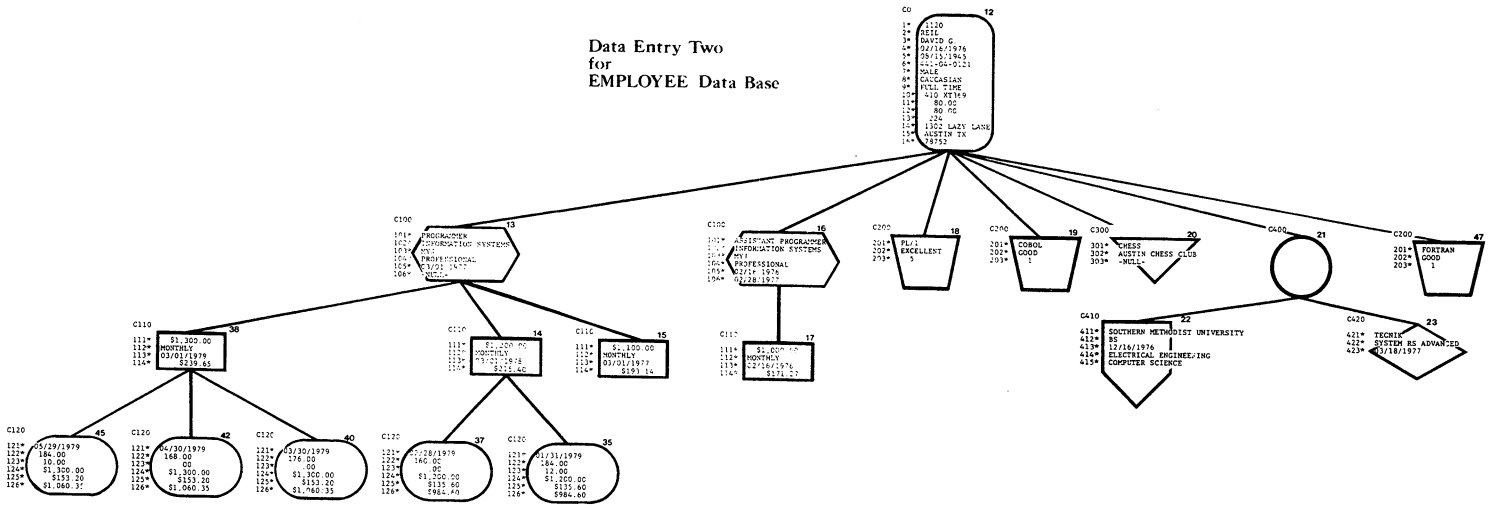
Data Base Schema for EMPLOYEE Data Base



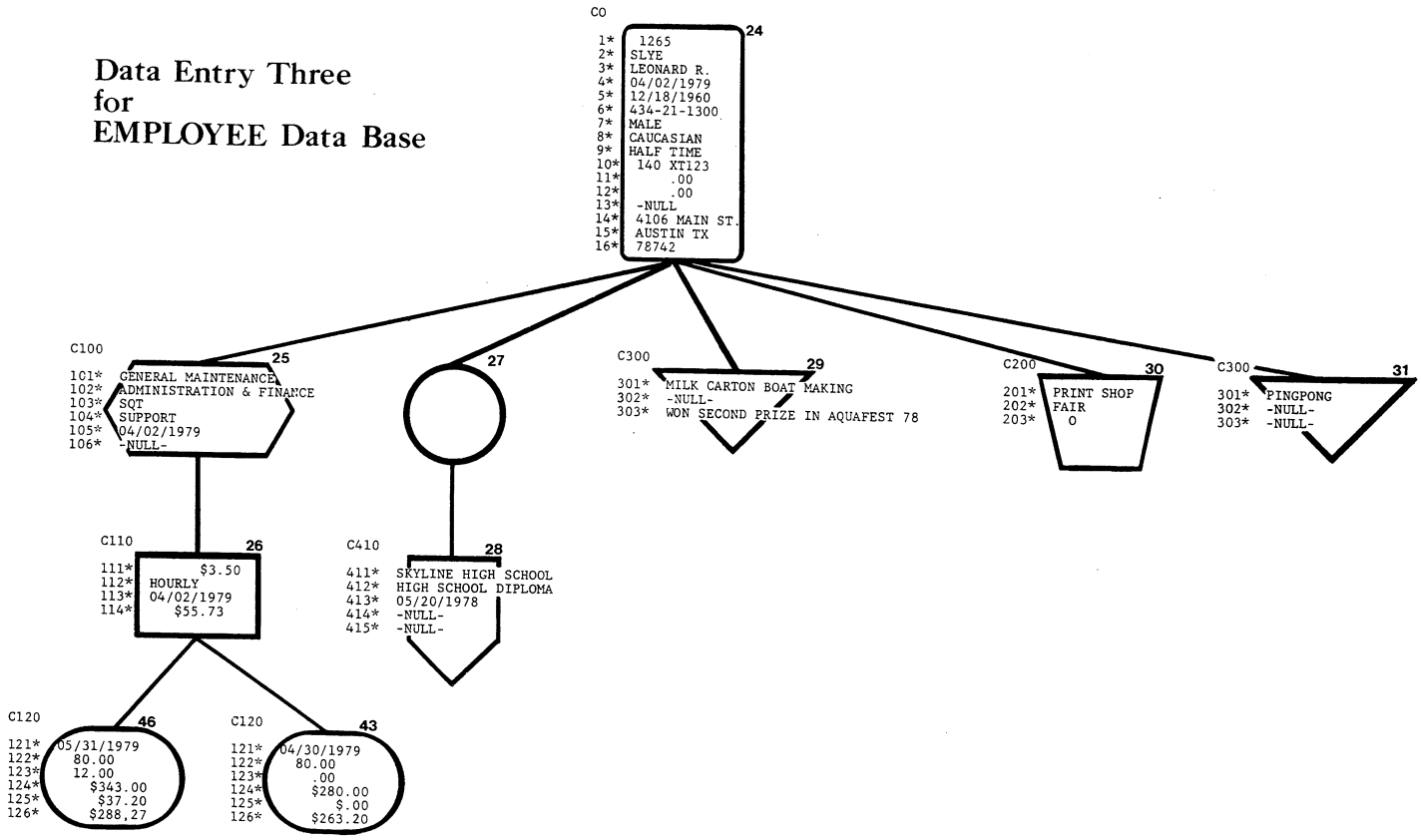
Data Entry One for EMPLOYEE Data Base



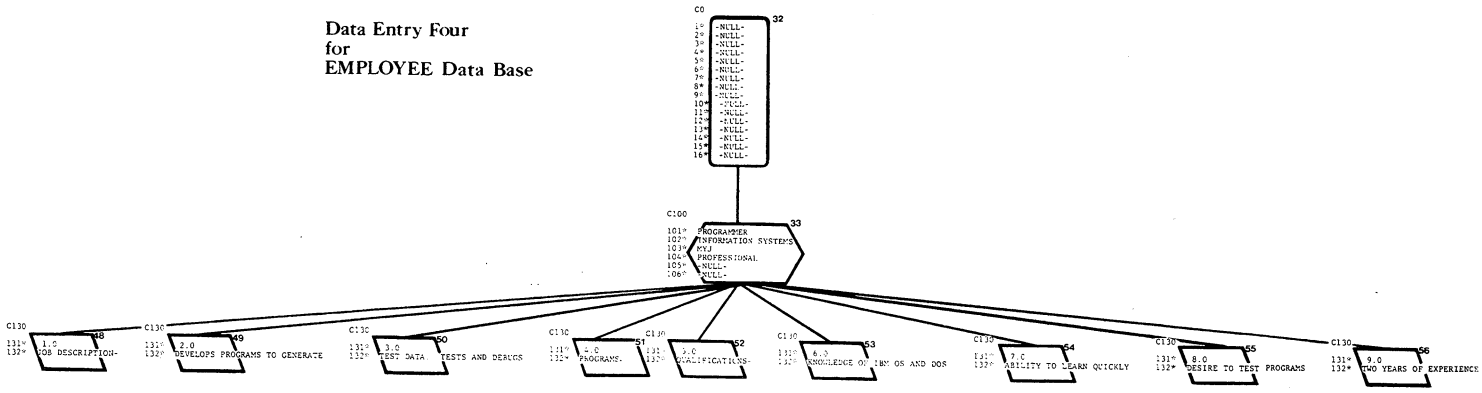
Data Entry Two
for
EMPLOYEE Data Base



Data Entry Three
for
EMPLOYEE Data Base



Data Entry Four
for
EMPLOYEE Data Base



Your Turn

If you have any comments about SYSTEM 2000® or this manual, please let us know by writing your ideas in the space below. If you include your name and address, we will reply to you.

Please return this sheet to Publications Division, P.O. Box 200075, Austin, TX 78720-0075

IMPORTANT Please include:
Manual Title: _____
Document Number: _____