DATATRIEVE-11

digital

User's Guide

Order Number: AA-X023C-TK

DATATRIEVE–11
User's Guide

Order Number: AA–X023C–TK

July 1989

This document describes how to use DATATRIEVE–11.

Operating Systems:	RSX–11M/M–PLUS RSTS/E Micro/RSX Micro/RSTS VMS with VAX–11 RSX
Software Version:	DATATRIEVE–11 Version 3.3

digital equipment corporation
maynard, massachusetts

First Printing, September 1983
Revised, November 1987
Revised, July 1989

The postpaid Reader's Comments forms at the end of this document request your critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DATATRIEVE	Rdb/VMS	VAX Information Architecture
DATATRIEVE–11	ReGIS	VAX Rdb/ELN
DEC	RSTS	VAXcluster
DECnet	RSTS/E	VAXinfo
DECUS	RSX	VAX/VMS
Micro/RSTS	RSX–11M	VAX–11 RSX
Micro/RSX	RSX–11M–PLUS	VMS
MicroVAX	UNIBUS	VT
MicroVMS	VAX	
PDP	VAX CDD	
PDP–11	VAX DATATRIEVE	digital™

ZK5063

Contents

Index

Figures

Tables

Preface

This manual is a guide to the interactive use of DATATRIEVE–11. It explains how to define dictionaries and dictionary objects (domains, records, tables, and procedures), and gives examples of using compound statements, command files, views, and hierarchies. It also discusses the restructuring of domains and the control of input and output.

In this manual, the DATATRIEVE–11 software is also referred to as DATATRIEVE.

Intended Audience

This manual is intended for people who:

- Have read and tried the examples in the *Introduction to DATATRIEVE–11*
- Have experience using DATATRIEVE
- Have experience in applications programming but are unfamiliar with DATATRIEVE

For people who have no experience with DATATRIEVE, the *Introduction to DATATRIEVE–11* provides a tutorial for the basic DATATRIEVE–11 tasks.

Structure

This manual is divided into 20 chapters, an appendix, and an index:

Chapter 1	Introduces the basic concepts of DATATRIEVE–11.

Chapter 2	Describes how to enter and exit DATATRIEVE, display data, use various prompts, and how to use DATATRIEVE Help and Guide Mode.
Chapter 3	Describes data dictionaries and how to create or change them.
Chapter 4	Describes how to name and define domains.
Chapter 5	Describes how to plan a record definition, the rules governing the definition, the optional clauses you can use to specify certain conditions, and the use of lists to define hierarchical records.
Chapter 6	Describes the difference between indexed and sequential data files, compares their advantages, and explains how to define them.
Chapter 7	Discusses record selection expressions (RSEs) and how to use them to select particular records from your database.
Chapter 8	Describes the use of REPEAT, FOR, and BEGIN-END blocks to combine statements into compound statements.
Chapter 9	Describes how to define and use DATATRIEVE procedures.
Chapter 10	Describes how to create and use DATATRIEVE command files.
Chapter 11	Discusses DATATRIEVE variables, the differences between local and global variables, and how to use them.
Chapter 12	Describes the creation and use of DATATRIEVE tables.
Chapter 13	Describes how to define and use view domains to examine only part of one domain or to combine information from more than one domain.
Chapter 14	Discusses how to use lists to retrieve data.
Chapter 15	Explains how to change record and file definitions to restructure a domain.
Chapter 16	Explains how to use the DATATRIEVE Editor.
Chapter 17	Discusses the most effective ways of using your allotted workspace.
Chapter 18	Describes ways to control the format of your output and how to use the SET ABORT and SET PROMPT commands.
Chapter 19	Describes how to create, use, and maintain access control lists.

Chapter 20	Describes how to modify and delete data dictionaries, optimize the disk storage space they use, and extract their contents.
Appendix A	Explains in detail the way DATATRIEVE establishes context, the context stacks, context variables, and single record context.

Related Manuals

For further information on the topics covered in this manual, refer to the following manuals:

- *Introduction to DATATRIEVE–11*
- *DATATRIEVE–11 Call Interface Manual*
- *DATATRIEVE–11 Reference Manual*

Conventions

This section explains the special symbols used in this book:

Convention	Meaning
RET	This symbol indicates the RETURN key.
TAB	This symbol indicates the TAB key.
>	The right angle bracket symbol at the beginning of a new line, or by itself, represents the system prompt.
Color	Color in examples shows user input.
UPPERCASE	Uppercase words represent DATATRIEVE keywords.
lowercase	Lowercase words are generic terms that indicate entries you must provide.
{ }	Braces enclose clauses from which you must choose one alternative.
[ ]	Square brackets enclose optional clauses from which you can choose one or none.
. . .	Horizontal ellipsis means you can repeat the previous item.
. . .	Vertical ellipsis in an example means that information not directly related to the example has been omitted.

Summary of Technical Changes

DATATRIEVE–11 Version 3.3 incorporates the following technical changes to the software:

- The EDIT command now invokes the EDT text editor. In previous versions, the EDIT command invoked the DATATRIEVE Query Editor (QED), also known as the DATATRIEVE Editor, which is no longer supported. The EDT language allows keypad editing of any DATATRIEVE object.
- The DATE data type has been expanded to represent time values.
- Seventeen lexical functions have been added to the DATATRIEVE language.
- Installation on RSX–11M/M–PLUS and RSTS/E systems now uses Auto-Install.

Chapter 1

Introduction to DATATRIEVE–11

This chapter gives an overview of DATATRIEVE–11 and explains its basic concepts and terms.

1.1 Description of DATATRIEVE–11

DATATRIEVE–11 is an interactive tool for inquiry, update, and maintenance of information stored in data files. DATATRIEVE–11 runs on any PDP–11 computer with an RSX–11M/M–PLUS, RSTS/E, Micro/RSX, or Micro/RSTS operating system. In addition, it runs on a VAX system with VAX–11 RSX installed under VMS.

The commands and statements you use are common English words that have specific meaning within DATATRIEVE. With these commands and statements, you can manipulate the information in selected data files.

1.2 DATATRIEVE Concepts and Terms

DATATRIEVE–11 uses the following concepts and terms:

- Files, records, and domains
- Data dictionaries
- Commands and statements
- Procedures
- Command files
- Tables

Each of these are discussed in the following sections.

1.2.1 Files

A data file contains ordered data. The DATATRIEVE DEFINE FILE command creates a data file and allows you to specify some of the characteristics of the file. You can use your record definition to control the way DATATRIEVE handles the data in other data files. You can also use a number of different record definitions to work with the information stored in one data file.

1.2.2 Record Definitions

A record is the basic unit of information management. It describes the relationship between logically connected data items and consists of one or more subunits called fields. DATATRIEVE uses both elementary and group fields. An elementary field contains an item of data. A group field contains group and elementary fields that are logically related. Thus, you might have an EMPLOYEE_NAME group field that contains the elementary fields FIRST_NAME, MIDDLE_INITIAL, and LAST_NAME.

DATATRIEVE enables you to interpret the data in the fields of a data record. The DATATRIEVE record definition controls the way you look at and interpret the information coded in your data files.

In the field definition clauses in a DEFINE RECORD command, you specify the type of field, the length of each field, and the content of the fields: character strings, numbers, or dates. You can also control the format DATATRIEVE uses to display values from those fields.

Note that in field names (like FIRST_NAME), this manual uses underscores (_). When you are using DATATRIEVE, you may use either underscores or hyphens (-) in record or field definitions. DATATRIEVE accepts hyphens as input but converts them to underscores before analyzing user input. To use a hyphen as a minus sign, put spaces before and after it. Otherwise, DATATRIEVE converts the minus sign to an underscore and issues an error message.

Be consistent in your own practice. To avoid confusion when using the manuals, you may want to use underscores instead of hyphens. Whichever you use, Chapter 5 of this manual explains record definitions and field definition clauses.

1.2.3 Domains

To manage the data in a file, you must explicitly connect that data file to a record definition. DATATRIEVE makes this connection with a domain. You use the DEFINE DOMAIN command to create a domain definition and to store that definition in your current data dictionary.

The domain definition establishes a name for the domain and associates that name with the names of a record definition and a data file. When you use the name of the domain, you tell DATATRIEVE to use a particular record definition to interpret the data stored in a specific file.

Do not confuse a domain with the data you want to manage. The data stored in the data file does not constitute the domain. You can erase old records and add new ones without disturbing the relationship between the data file and the record definition, which remains fixed.

1.2.4 Data Dictionaries

A data dictionary is an RMS file that DATATRIEVE creates to store definitions. It stores tables, procedures, and the definitions of domains and records.

1.2.5 Commands and Statements

You manage information with commands and statements based on the DATATRIEVE keywords described in this manual.

Commands deal with the data dictionary and enable you to:

- Create, change, and display definitions in the dictionary (DEFINE, SHOW, EDIT, DELETE and EXTRACT)
- Maintain the access control lists associated with those definitions (DEFINEP, SHOWP, and DELETEP)
- Get access to domains (READY)
- Release access to domains, variables, collections, and DATATRIEVE tables (RELEASE and FINISH)
- Determine the way DATATRIEVE displays data on your terminal (SET)
- Get online information about DATATRIEVE (HELP)
- End your DATATRIEVE sessions (EXIT)

Statements deal with data and enable you to:

- Store records (STORE)
- Form and manipulate groups of records (FIND, SELECT, DROP, and SORT)
- Display data (PRINT, REPORT, and SUM)
- Modify records (MODIFY)
- Erase records (ERASE)
- Declare variables and assign values (DECLARE and assignment)

You can join statements to form compound statements in the BEGIN-END, FOR, IF-THEN-ELSE, REPEAT, and THEN statements. You cannot join commands with other commands or with statements. Furthermore, only statements may contain DATATRIEVE value expressions and record selection expressions.

You can enter statements at levels other than command level (indicated by the DTR> prompt), but you can enter commands at command level only.

If you are not sure whether a given keyword is a command or a statement, refer to the table showing an alphabetical summary of commands and statements in the *DATATRIEVE–11 Reference Manual.*

1.2.6 Procedures

Most DATATRIEVE–11 applications involve sequences of commands and statements that you use again and again. To avoid retyping such a sequence you can store it in your dictionary as a procedure. You can also use procedures within commands or statements. With the DEFINE PROCEDURE command, you give the sequence a name and enter both the name and the sequence into your dictionary. You invoke the procedure by entering a colon (:) and the procedure name. DATATRIEVE then interprets the content of the procedure just as though you had entered it at your terminal.

1.2.7 Command Files

You can use command files in DATATRIEVE in much the way you use DATATRIEVE procedures. The primary difference between the two is that you store command files in your operating system directory and procedures in a DATATRIEVE dictionary.

When you invoke a command file, DATATRIEVE displays the text in the command file on your terminal. When you invoke a procedure, however, the procedure definition is not displayed on your terminal.

1.2.8 DATATRIEVE Tables

DATATRIEVE tables perform two functions. They let you:

- Specify one value and retrieve another that you have associated with the first
- Validate data according to the presence or absence of a data item in the table

A table contains pairs of character strings. The first member of each pair is the code string and the second is the translation string. The last entry in the table can be an ELSE clause that specifies a default translation string. Thus your table might match department codes (such as A01) with department names (such as Accounting). For example, if you enter a code not in the table, the ELSE clause can specify a message to be displayed on your terminal, telling you the code is not valid.

1.3 Components of DATATRIEVE–11

DATATRIEVE–11 consists of four components on your PDP–11 system:

- Interactive DATATRIEVE–11

  The DTR.TSK task image allows you to access DATATRIEVE at your terminal.

- The DATATRIEVE–11 Distributed Server

  DDMF.TSK allows users on other DECnet nodes to use DATATRIEVE for accessing data files and data dictionaries on your node.

- The DATATRIEVE–11 Call Interface

  The DTCLIB.OLB object module library allows application programs in other high-level languages to call DATATRIEVE subroutines. The calls, through a local or remote server, give you access to DATATRIEVE data files and dictionaries.

- The DATATRIEVE–11 Remote Terminal Interface

  REMDTR.TSK is an interactive program that uses the Call Interface and the remote server. When you run REMDTR as a program, it looks as though you are running interactive DATATRIEVE on a remote node.

The following sections describe these four components and tell you where to find information on each one.

1.3.1 Interactive DATATRIEVE–11

When you invoke DATATRIEVE–11 on a PDP–11 system, you are running DTR.TSK, the interactive DATATRIEVE task image. This program accepts DATATRIEVE commands and statements from the terminal and uses the terminal as the default output device. DTR.TSK allows you to access data stored in disk files as well as definitions stored in one of the data dictionaries on your system, using DATATRIEVE commands and statements. The *Introduction to DATATRIEVE–11*, the *DATATRIEVE–11 Reference Manual*, and this manual describe how to use interactive DATATRIEVE.

1.3.2 The DATATRIEVE–11 Distributed Server

The Distributed Data Manipulation Facility (DDMF) is also called the DATATRIEVE Distributed Server. It is a "slave" program. Another DATATRIEVE component sends it commands to execute, and it passes the results back to that component. DDMF can perform all the DATATRIEVE functions that DTR.TSK can perform, with the exception of the Application Design Tool (ADT) and Guide Mode.

You use the Distributed Server in three ways:

- VAX DATATRIEVE uses the Distributed Server on a PDP–11 or VAX node to perform distributed operations. For example, when you type READY YACHTS AT FRODO in VAX DATATRIEVE, DATATRIEVE starts up DDMF on the node named FRODO and uses it to access definitions and data files on that node.
- The DATATRIEVE–11 Call Interface uses DDMF to give you access to data dictionaries and data files, allowing you to write application programs that call DATATRIEVE.
- The DATATRIEVE–11 Remote Terminal Interface uses DDMF through the Call Interface, allowing you to use your terminal to access DATATRIEVE on other nodes.

1.3.3 The DATATRIEVE–11 Call Interface

The DATATRIEVE–11 Call Interface allows you to write high-level language programs that call DATATRIEVE, either on your own system or on another DECnet node. To use the Call Interface, you include calls to external DATATRIEVE subroutines in your program. When you build the task image, you link the program to the object module library DTCLIB.OLB. The subroutines pass information between the calling program and a local or remote DATATRIEVE Distributed Server. When you are running such a program, there are actually two task images active:

- Your program linked to DTCLIB.OLB
- DDMF, the DATATRIEVE Distributed Server that has been activated to serve your program

NOTE

You must have the DECnet software installed on your system before you can access DATATRIEVE on a remote node.

The *DATATRIEVE–11 Call Interface Manual* tells you how to write programs that use the Call Interface.

1.3.4 The DATATRIEVE–11 Remote Terminal Interface

The DATATRIEVE–11 Remote Terminal Interface (REMDTR) gives you interactive access to DATATRIEVE on other nodes.

NOTE

You must have the DECnet software installed on your system before you can use REMDTR.

To use the Remote Terminal Interface on both RSTS and RSX–11M/M–PLUS systems, type RUN $REMDTR. If an error message is displayed, check with your system manager to make sure the program is installed.

When REMDTR prompts you for a node name, you can type either a node name or a complete network address specification. The address specification includes a user name or account number and a password:

```
Enter node name: MYVAX
Enter node name: MYVAX"MYNAME PASWRD"
Enter node name: MYRSTS"130,34 PASWRD"
```

If you specify only the node name, you are logged in to the default DECnet account and may not have access to the data files or dictionaries you want. When you type the complete form of the specification, DECnet logs you into that account.

After you have logged in successfully, you can use DATATRIEVE interactively on that node as if you were using DATATRIEVE on your own node, except that Guide Mode and ADT are not available.

You can use the Remote Terminal Interface for copying dictionary definitions and data files across DECnet. It is also useful for testing a network path and determining the default characteristics of DDMF, the DATATRIEVE Distributed Server, on a remote node.

For more information on using the Remote Terminal Interface, see the *DATATRIEVE–11 Call Interface Manual.*

Chapter 2

Getting Started with DATATRIEVE

This chapter tells you how to start and stop DATATRIEVE. It also tells you how to:

- Work with sample domains, records, and files included in the DATATRIEVE–11 installation kit
- Use the PRINT and REPORT statements
- Understand DATATRIEVE prompts
- Interpret DATATRIEVE error messages
- Use Help and Guide Mode

2.1 Invoking DATATRIEVE–11

The way you start DATATRIEVE can vary from one system to another. If you cannot invoke DATATRIEVE using the method discussed in this section, contact the person in charge of DATATRIEVE on your system.

See your system manager for the exact invocation line for your system. Here is how to start the system if your invocation line is RUN $DTR:

```
> RUN $DTR RET
PDP-11 DATATRIEVE, DEC Query and Report System
Version: V3.3-nn, 13-JUN-89
Type HELP for help
DTR>
```

The startup banner in the previous example shows you that you have successfully invoked DATATRIEVE.

2.2 Sample Domains, Records, and Data Files

The DATATRIEVE–11 installation kit includes four sample domains: YACHTS, OWNERS, PERSONNEL, and FAMILIES. The domain definitions and the record definitions are in the system data dictionary.

The following command creates a private dictionary for you called SAMPLE.DIC which resides in your current default directory. The command enters the domain and record definitions into the dictionary, and copies the data files into your directory.

For RSTS/E and Micro/RSTS systems type the following:

```
DTR> @LB:SETUP.DTR RET
DTR>
```

For RSX, Micro/RSX, and VAX–11 RSX systems type the following:

```
DTR> @LB:[1,2]SETUP.DTR RET
DTR>
```

To see that the sample domain and record definitions are in place, use the SHOW command:

```
DTR> SHOW DOMAINS, RECORDS RET
Domains:
        FAMILIES        KETCHES         OWNERS          OWNERS_SEQUENTIAL
        PERSONNEL       PERSONNEL_SEQ   SAILBOATS       YACHTS
        YACHTS_SEQUENTIAL
Records:
        OWNER_RECORD    PERSONNEL_REC   PERSONNEL_SEQ_REC
        YACHT
DTR>
```

The results of this command vary from one system to another, but you should be sure that DATATRIEVE lists the needed domain and record definitions on your terminal. If you have difficulty, see the person responsible for DATATRIEVE–11 on your system.

2.3 QUERY.INI Startup File

If you frequently start your DATATRIEVE session with the same series of commands and statements, you can use a command file to execute the commands and statements automatically each time you invoke DATATRIEVE. DATATRIEVE recognizes QUERY.INI as the default startup file. However, you can specify a different startup file at installation time if you choose.

DATATRIEVE first looks for a QUERY.INI file in your default directory. If one is there, DATATRIEVE executes the commands and statements it contains before it accepts any other input.

If this file cannot be found in the current directory, DATATRIEVE searches for QUERY.INI in the same location as QUERY.DIC. This allows for a system-generated QUERY.INI as a default if the user does not provide QUERY.INI. Recommended file protection of the system QUERY.INI file is World Read.

If you would like to be in Guide Mode as soon as you invoke DATATRIEVE, for instance, you can include the SET GUIDE comand at the end of your QUERY.INI file. You can put a SET DICTIONARY command in the QUERY.INI file to automatically change your default data dictionary. You can ready domains you use frequently. Your QUERY.INI file, then, might include the following lines:

```
SET DICTIONARY MYDIC.DIC
SET GUIDE
READY YACHTS
```

When you type your command to invoke DATATRIEVE, you are placed in your own dictionary and in Guide Mode, and the YACHTS domain is readied.

2.4 Using READY, PRINT, and REPORT to Retrieve Data

The basic keywords used to retrieve data are the READY command and the PRINT and REPORT statements. To display a complete domain, for example, first ready the domain. Then type PRINT followed by the domain name:

```
DTR> READY PERSONNEL RET
DTR> PRINT PERSONNEL RET
```

```
                    FIRST       LAST                START              SUP
 ID     STATUS      NAME        NAME      DEPT      DATE      SALARY   ID

00012 EXPERIENCED CHARLOTTE   SPIVA       TOP   12-Sep-72   $75,892 00012
00891 EXPERIENCED FRED        HOWL        F11   9-Apr-76    $59,594 00012
02943 EXPERIENCED CASS        TERRY       D98   2-Jan-80    $29,908 39485
12643 TRAINEE     JEFF        TASHKENT    C82   4-Apr-81    $32,918 87465
32432 TRAINEE     THOMAS      SCHWEIK     F11   7-Nov-81    $26,723 00891
34456 TRAINEE     HANK        MORRISON    T32   1-Mar-82    $30,000 87289
38462 EXPERIENCED BILL        SWAY        T32   5-May-80    $54,000 00012
38465 EXPERIENCED JOANNE      FREIBURG    E46   20-Feb-80   $23,908 48475
39485 EXPERIENCED DEE         TERRICK     D98   2-May-77    $55,829 00012
48475 EXPERIENCED GAIL        CASSIDY     E46   2-May-78    $55,407 00012
48573 TRAINEE     SY          KELLER      T32   2-Aug-81    $31,546 87289
49001 EXPERIENCED DAN         ROBERTS     C82   7-Jul-79    $41,395 87465
49843 TRAINEE     BART        HAMMER      D98   4-Aug-81    $26,392 39485
78923 EXPERIENCED LYDIA       HARRISON    F11   19-Jun-79   $40,747 00891
83764 EXPERIENCED JIM         MEADER      T32   4-Apr-80    $41,029 87289
84375 EXPERIENCED MARY        NALEVO      D98   3-Jan-76    $56,847 39485
87289 EXPERIENCED LOUISE      DEPALMA     G20   28-Feb-79   $57,598 00012
87465 EXPERIENCED ANTHONY     IACOBONE    C82   2-Jan-73    $58,462 00012
87701 TRAINEE     NATHANIEL   CHONTZ      F11   28-Jan-82   $24,502 00891
88001 EXPERIENCED DAVID       LITELLA     G20   11-Nov-80   $34,933 87289
90342 EXPERIENCED BRUNO       DONCHIKOV   C82   9-Aug-78    $35,952 87465
91023 TRAINEE     STAN        WITTGEN     G20   23-Dec-81   $25,023 87289
99029 EXPERIENCED RANDY       PODERESIAN  C82   24-May-79   $33,738 87465

DTR>
```

For an explanation of the various forms of the PRINT statement, see the *Introduction to DATATRIEVE–11* and the *DATATRIEVE–11 Reference Manual*. To retrieve information in a report format, use the REPORT statement. This can provide you with a report title, a date, page numbers, and various statistical functions. In its simplest form, the report specification consists of a REPORT statement, followed by a PRINT statement specifying the fields you want to report, and an END_REPORT statement to conclude the specification:

```
DTR> REPORT YACHTS WITH BUILDER = "ALBERG" RET
RW> PRINT BOAT RET
RW> END_REPORT RET

                15-Nov-87
                  Page 1

                                          LENGTH
                                           OVER
  MANUFACTURER       MODEL        RIG      ALL     WEIGHT    BEAM     PRICE

    ALBERG         37 MK II      KETCH     37      20,000     12     $36,951

DTR>
```

For a complete explanation of the DATATRIEVE–11 Report Writer, see the *DATATRIEVE–11 Guide to Writing Reports*.

2.5 DATATRIEVE Input Line Prompts

Several types of prompts provide you with information during your interactive DATATRIEVE session. There are four types of input line prompts:

DTR>	Marks the beginning of input lines and shows that DATATRIEVE is ready for your input
CON>	Prompts you to continue incomplete commands or statements and shows what DATATRIEVE expects next
DFN>	Prompts you to continue a partially complete DEFINE command
RW>	Prompts you to complete unfinished Report Writer statements and enter additional statements

2.6 DATATRIEVE Syntax Line Prompts

When you press RETURN before completing a command or statement, DATATRIEVE prompts you with a phrase in square brackets telling you what sort of input it expects to satisfy the syntax of the command or statement. The following example shows several of DATATRIEVE's syntax prompts:

```
DTR> READY RET
[Looking for Dictionary Element]
CON> YACHTS RET
DTR> FIND RET
[Looking for "FIRST", domain name, or collection name]
CON> YACHTS WITH RET
[Looking for Boolean expression]
CON> LOA RET
[Looking for relational operator (eq, gt, etc.)]
CON> BETWEEN RET
[Looking for a value expression]
CON> 20 RET
[Looking for upper value of between]
CON> AND RET
[Looking for a value expression]
CON> 30 RET
[54 records found]
DTR>
```

2.7 DATATRIEVE Prompts for Storing and Modifying Values

When you enter a STORE or MODIFY statement, DATATRIEVE usually prompts you to enter new information to be stored in the record, or to replace what was stored there previously. Unless the statement contains a USING clause, you are prompted to enter information for each field receiving a new value. The prompt is made up of the word "Enter" followed by the name of the field. It takes the following general form:

```
Enter field-name:
```

2.8 DATATRIEVE Error Messages

Error messages are DATATRIEVE responses to faulty syntax or an error in logic. When it detects an error, DATATRIEVE displays an error message and returns you to DATATRIEVE command level. All data items remain as they were before you made the error. The messages describe the error. For instance, when you use an undefined name in a PRINT command, DATATRIEVE responds with an error message:

```
DTR> PRINT RUBINS RET
Field "RUBINS" is undefined or used out of context
DTR>
```

2.9 Ending Your DATATRIEVE Session

You can exit from DATATRIEVE by doing one of the following:

- Enter the EXIT command
- Press CTRL/Z
- Press CTRL/C two times

2.9.1 Using the EXIT Command

To end your DATATRIEVE session and return to the operating system command level, you can respond to the DTR> prompt with an EXIT command:

```
DTR> EXIT RET
>
```

Entering the EXIT command at any other DATATRIEVE prompt is a syntax error.

2.9.2 Exiting from DATATRIEVE with CTRL/Z

You can also end your DATATRIEVE session by entering CTRL/Z in response to the DTR> prompt:

```
DTR> CTRL/Z
>
```

Entering CTRL/Z in response to any prompt other than DTR> returns you to DATATRIEVE command level (DTR>).

2.9.3 Exiting from DATATRIEVE with CTRL/C

If you enter one CTRL/C when DATATRIEVE is executing a command, DATATRIEVE aborts its operation. If you enter CTRL/C two consecutive times, the second aborts the task and returns you to your operating system command level. Using either the EXIT command or CTRL/Z is a better way to exit than using two CTRL/Cs, because EXIT and CTRL/Z do not abort the task.

2.10 Using Help

DATATRIEVE offers two levels of help: basic and advanced. Basic help is information about elementary DATATRIEVE statements. Advanced help provides instructions on statements that you are likely to use after some experience with DATATRIEVE. To get information on the help topic itself, type HELP in response to the DTR> prompt.

For a listing of the topics for which help is available, type:

```
DTR> HELP HELP RET

The topics of greatest interest to the  beginning  user  are
the following:

EXIT            FIND            GUIDE           MODIFY
PRINT           READY           SORT            STORE
```

```
Help is available for the following topics:

ABORT           ADT             ASSIGNMENT      BEGIN
COMPUTED        CONDITION       DATE            DECLARE
DEFINE          DEFINEP         DELETE          DELETEP
DICTIONARY      DISPLAY         DOMAIN          DROP
EDIT            EDIT-STRING     ERASE           EXIT
EXTRACT         FILE            FIND            FINISH
FN$ABS          FN$DATE         FN$DAY          FN$DCL
FN$HOUR         FN$HUNDREDTH    FN$JULIAN       FN$MINUTE
FN$MOD          FN$MONTH        FN$SECOND       FN$SIGN
FN$STR_EXTRACT  FN$STR_LOC      FN$TIME         FN$UPCASE
FN$YEAR         FOR             GUIDE           HELP
IF              LEXICALS        MODIFY          OCCURS
PIC             PRINT           PROCEDURE       READY
RECORD          RELEASE         REPEAT          REPORT
RSE             SELECT          SET             SHOW
SHOWP           SORT            STORE           SUM
TABLE           THEN            USAGE           VALUE
VIEW

DTR>
```

To get help on one of these topics, type HELP followed by the name of the topic and press the RETURN key.

For a listing of topics for which advanced help is available, type:

```
DTR> HELP ADVANCED HELP RET
Advanced help is available for the following topics:

CONDITION       DICTIONARY      DOMAIN          EDIT
EDIT-STRING     FILE            FIND
MODIFY          OCCURS          PRINT           PROCEDURE
RECORD          RSE             SELECT          SORT
STORE           TABLE           USAGE           VALUE
VIEW

DTR>
```

To get advanced help on one of these subjects, type HELP ADVANCED followed by the name of the subject.

2.11 Guide Mode

The self-explanatory Guide Mode feature helps you:

- While you are learning to use DATATRIEVE
- Whenever you are unsure of the sequence of commands you need to accomplish your task

NOTE

To use Guide Mode, you must have a video display terminal. Guide Mode does not work on any hardcopy terminal.

To get into Guide Mode, type:

```
DTR> SET GUIDE RET
```

DATATRIEVE then prompts you, step-by-step, for every operation you want to perform. If you need help, type a question mark (?). DATATRIEVE shows you a list of the possible commands and statements you can use to complete your task:

```
DTR> SET GUIDE RET
 Enter command, type ? for help
```

If you type ?, DATATRIEVE displays the following information:

```
Enter command, type ? for help

The possible responses are:

    READY           Make domain available
    SHOW            Display status information
    LEAVE           Return to normal Datatrieve
```

Notice that Guide Mode automatically spells out entire words and phrases immediately after you type only one or two letters. You may find this somewhat startling at first, but you soon get used to it. Guide Mode fills in the word as soon as you enter characters it recognizes. If you type R, it completes the word as READY. If you type RE, REA, and so on, it also echoes the word READY.

You can get unexpected results with Guide Mode if you are not careful, however. For example, suppose you want to ready a domain called EASELS. If you type R EA, Guide Mode does not expand the R to READY and the EA to EASELS. Rather, it interprets the EA as part of a READY command with no domain name supplied. To get the results you want, you could instead type REA EA, which would be expanded to READY EASELS.

When you are ready to exit from Guide Mode and return to regular DATATRIEVE, type LEAVE. The DTR> prompt indicates you are out of Guide Mode.

Chapter 3

Using Data Dictionaries

Invoking DATATRIEVE automatically connects you to a data dictionary: an RMS file that DATATRIEVE creates to store definitions and protection information. This chapter shows how to create and use a data dictionary.

3.1 Contents of a Data Dictionary

A data dictionary contains the definitions and protection information for the following DATATRIEVE data structures:

- Domains
- Records
- Procedures
- Description tables

Each definition describes the contents of the data structure:

- A domain definition contains the name of the domain, the name of the record definition associated with the domain, and the file specification of the file containing the data for the domain.
- A record definition contains the name of the record and a definition for each field in the record.
- A procedure definition is the procedure itself, including the procedure name and all commands and statements in the procedure.
- A table definition is the table itself, including its name and all code and description pairs.

Associated with each of these definitions is an access control list (ACL). It protects the one definition with which it is associated and restricts its use. Access control lists supplement the protection features of your operating system.

3.2 Creating a Data Dictionary

You create a data dictionary using the DEFINE DICTIONARY command. The format of this command is:

```
DEFINE DICTIONARY  file-spec
```

DEFINE DICTIONARY creates an empty indexed sequential RMS file that is suitable for use as a data dictionary. You supply the file specification, and the operating system creates an entry for it in your directory. You can choose any file extension when you create it. If you use the default file extension .DIC, you do not have to specify the file extension when referring to your dictionary file from DATATRIEVE command level. In addition, if you use the standard file extension, you can easily identify your dictionary files when you list your directory.

When you enter a DEFINE DICTIONARY command, DATATRIEVE creates a file and establishes the new dictionary as your current data dictionary, just as if you had entered a SET DICTIONARY command:

```
DTR> SHOW DICTIONARY RET
The current dictionary is SY:[1,3]NEWQ.DIC
DTR> DEFINE DICTIONARY MYDIC RET
DTR> SHOW DICTIONARY RET
The current dictionary is SY:[1,37]MYDIC.DIC
```

Note that because you did not specify any extension for the dictionary, DATATRIEVE assigned .DIC by default.

You can begin entering definitions immediately. If DATATRIEVE cannot create a file because, for example, you do not have write access to the disk, or the disk is not mounted, it prints a message on your terminal and leaves you connected to your current dictionary.

3.3 Changing Dictionaries

When you invoke it, DATATRIEVE automatically connects you to a default data dictionary. At the time of installation, your system manager determines what the default dictionary will be for your system. At the beginning of your DATATRIEVE session, that dictionary is your current dictionary.

To find out the name of your current dictionary, use the following command:

```
DTR> SHOW DICTIONARY RET
The current dictionary is SY:[1,37]MYDIC.DIC
```

SHOW DICTIONARY prints the file specification of the current data dictionary.

If you want to use a different dictionary, use the following format of the SET command:

```
SET DICTIONARY file-spec
```

To change from the default dictionary to another dictionary, you must specify at least one element of the file specification of the other data dictionary. For example, if you specify only the device name, DATATRIEVE searches that device for a directory with your project-programmer number (PPN) or user identification code (UIC) and a file named QUERY.DIC. If DATATRIEVE does not find the file you specify, it prints an error message on your terminal and leaves you in your current dictionary. (At no time are you in DATATRIEVE without being in a data dictionary.)

If you have readied a domain in your current dictionary and you change dictionaries, that domain is still available when you are in the new dictionary. This feature allows you to move records from that readied domain into another domain in the new current dictionary.

If you want to return to the default data dictionary, issue the SET DICTIONARY command without a file specification:

```
DTR> SET DICTIONARY RET
```

The following dialogue illustrates the setting and displaying of data dictionaries. Note that if you try to set a dictionary that you have not defined, DATATRIEVE prints an error message on your terminal. You must first define the dictionary with a DEFINE DICTIONARY command. Note also that DATATRIEVE returns the DFN> prompt when you omit the file specification from the DEFINE DICTIONARY command. In this example, DR1: represents the name of the device on which the default DATATRIEVE system dictionary resides. DR2: is the name of the device on which your file directory is stored:

```
DTR> SHOW DICTIONARY RET
The current dictionary is DR1:[1,2]QUERY.DIC
DTR> SET DICTIONARY NEWDIC RET
File "NEWDIC" not found
DTR> DEFINE DICTIONARY RET
DFN> NEWDIC RET
DTR> SHOW DICTIONARY RET
The current dictionary is DR2:[200,200]NEWDIC.DIC
DTR> SET DICTIONARY RET
DTR> SHOW DICTIONARY RET
The current dictionary is DR1:[1,2]QUERY.DIC
DTR>
```

Note that you do not need an explicit SET DICTIONARY command after entering DEFINE DICTIONARY NEWDIC, because that DEFINE sets the dictionary to NEWDIC automatically.

Chapter 4

Defining Domains

When you define a DATATRIEVE domain, the domain associates the names of a record definition and a data file with each other. When you use a domain name, you tell DATATRIEVE to use a particular record definition to interpret the data stored in a specific file.

Do not confuse the domain with the data you want to manage using the domain. You can erase old records or add new ones to a data file without disturbing the relationship between the file and a record definition. That relationship is established when you define a domain and remains fixed.

This chapter explains how to define simple RMS domains. You can also use the Application Design Tool (ADT) to define domains, records, and files. For information on using ADT, see the *Introduction to DATATRIEVE–11*.

4.1 Specifying Domain Names

In the DEFINE DOMAIN command, you specify the name of a domain, the name of the record definition associated with the domain, and the file specification of the file containing the data for the domain. The domain name must:

- Begin with a letter (A–Z)
- Contain 31 characters or less
- End with a letter (A–Z) or a digit (0–9)
- Contain only letters, digits, dollar signs, hyphens, or underscores (A–Z, 0–9, $, -, or _)

DATATRIEVE enters the domain definition in your current dictionary directory.

NOTE

DATATRIEVE treats hyphens and underscores as identical characters. You may use either underscores or hyphens in the names you assign. When processing, DATATRIEVE automatically converts hyphens to underscores. When it returns the output, it shows underscores whether you have entered hyphens or underscores. To use a hyphen as a minus sign, put spaces before and after it.

4.2 Defining a Simple RMS Domain

To define a domain, you associate the name of the domain with a record definition and file specification. The format for the DEFINE DOMAIN command follows:

DEFINE DOMAIN domain-name USING record-name [(passwd) | (*)] ON file-spec

The domain name cannot duplicate a DATATRIEVE keyword or the name of any other element in the current data dictionary. The record name refers to the record definition to be associated with the domain. You can define the domain before you define the record. (See Chapter 5 for information on defining records.)

An optional password can be used to check for E (execute) privilege for the record definition. If an asterisk prompt (*) is specified, DATATRIEVE prompts you for the password.

Be sure to end the definition with a semicolon (;). If you omit the semicolon, DATATRIEVE prompts you for one with DFN>.

The following rules apply to the DEFINE DOMAIN command:

- It must be preceded by a DATATRIEVE command level prompt DTR>.
- You cannot include a domain definition in a procedure.
- You cannot invoke a procedure in a domain definition.
- You cannot include a DEFINE DOMAIN command in a DATATRIEVE statement.

The following is an example of the DEFINE DOMAIN command:

```
DTR> DEFINE DOMAIN SCHEDULE USING SCHED_REC ON SCHED; RET
DTR>
```

This command enters the definition for SCHEDULE in your current dictionary. The domain uses a record definition called SCHED_REC and a data file called SCHED.DAT. The file extension .DAT is assigned to the data file by default.

4.3 Using the SHOW Command with Domains

You can use the SHOW command to see how a domain is defined. If you enter SHOW followed by the domain name, the text you used in the DEFINE DOMAIN command is displayed on your terminal. For example, enter SHOW SCHEDULE to see how the domain SCHEDULE is defined:

```
DTR> SHOW SCHEDULE RET
DOMAIN SCHEDULE
 USING SCHED_REC ON SCHED;
DTR>
```

Note that the domain must be defined in your current default dictionary. (See Chapter 3 for information on dictionaries.)

To see if a domain is in your dictionary, enter the SHOW DOMAINS command. This displays a listing of all domains in your default dictionary, as follows:

```
DTR> SHOW DOMAINS RET
Domains:
        FAMILIES        KETCHES         OWNERS
        OWNERS_SEQUENTIAL               SAILBOATS       SCHEDULE
        YACHTS                          YACHTS_SEQUENTIAL
DTR>
```

Chapter 5

Defining Records

There are two ways to define a record in DATATRIEVE–11. You can use the interactive Application Design Tool (ADT), and DATATRIEVE creates the record for you based on your responses to a series of questions. You can also define the record yourself with the DEFINE RECORD command.

For simple records, the Application Design Tool is often an efficient way to complete your definition. See the *Introduction to DATATRIEVE–11* for a sample ADT session. The DEFINE RECORD command, on the other hand, lets you use options not available through ADT. For instance, it lets you include clauses such as the COMPUTED BY clause, which asks DATATRIEVE to calculate the value of a field from the values of other fields or value expressions.

This chapter explains how to set up a record definition using the DEFINE RECORD command. The *DATATRIEVE–11 Reference Manual* contains additional information on defining records, including alphabetical listings of all clauses you can use in your record definitions.

5.1 Planning a Record Definition

The first step in writing a DATATRIEVE record definition is to analyze your data. Decide what data items you need to manage, their relative importance, and ways to group related items.

You might want to maintain your personnel files with DATATRIEVE, for example. As you analyze the information, you find that for each employee you want to include an identification number, status (experienced or trainee), name, department, starting date, salary, and supervisor identification number. Your list of data items might look like that in Figure 5–1.

Figure 5–1: Data Items in a Personnel Record

IDENTIFICATION NUMBER
STATUS
EMPLOYEE NAME
DEPARTMENT
START DATE
SALARY
SUPERVISOR'S IDENTIFICATION NUMBER

ZK-0971A-HC

With your DATATRIEVE installation kit, you receive a record called PERSONNEL_REC made up of the data items listed in Figure 5–1. The DATATRIEVE definition for that record appears in Figure 5–2.

Figure 5–2: PERSONNEL_REC Record Definition

```
DTR> SHOW PERSONNEL_REC RET
RECORD PERSONNEL_REC
USING
01  PERSON.
    05 ID                      PIC IS 9(5).
    05 EMPLOYEE_STATUS         PIC IS X(11)
                               QUERY_NAME IS STATUS
                               QUERY_HEADER IS "STATUS"
                               VALID IF STATUS EQ "TRAINEE","EXPERIENCED".
    05 EMPLOYEE_NAME           QUERY_NAME IS NAME.
       10 FIRST_NAME               PIC IS X(10)
                                  QUERY_NAME IS F_NAME.
       10 LAST_NAME                PIC IS X(10)
                                  QUERY_NAME IS L_NAME.
    05 DEPT                    PIC IS XXX.
    05 START_DATE              USAGE IS DATE.
    05 SALARY                  PIC IS 9(5)
                               EDIT_STRING IS $$$,$$$.
    05 SUP_ID                  PIC IS 9(5).
;
DTR>
```

To write a DATATRIEVE record definition yourself, you provide the elements to transform a list of data items like that in Figure 5–1 into a formal record definition like that in Figure 5–2. The rest of this chapter tells you how to make such a change.

5.2 Getting Started and Naming a Record

When you define a record, you begin by specifying the name of a record. You can define the record interactively by entering the DEFINE RECORD command at the DTR> prompt followed by the complete record definition. If you make a mistake, DATATRIEVE displays an error message and returns you to the DATATRIEVE command level without saving the definition. You must then retype it.

To avoid retyping, you can define the record in a procedure (see Chapter 9) or a command file (see Chapter 10). You can define your record, or only a few fields of the record, complete the definition (remember the semicolon), and then revise it later. To add additional fields or make whatever other changes you would like, you can use the EDIT command or your preferred text editor to make the changes in a command file.

To make changes with the EDIT command:

1. Issue the EDIT command at the DTR> prompt, specifying the record that you want to change:

   ```
   DTR> EDIT record-name
   ```

2. Use EDT to make corrections. See Chapter 16 for a description of the EDIT command. See EDT documentation for full information about the EDT editing language.
3. When you exit the EDT command level, DATATRIEVE invokes the edited definition automatically.

To make the changes with your preferred text editor in a command file:

1. Copy the record definition to a file, using the EXTRACT statement as follows:

   ```
   EXTRACT ON file-spec record-name
   ```

2. EXIT from DATATRIEVE.
3. Use your text editor to make corrections in the record definition.

   Notice the file begins with the commands DELETE record-name and DEFINE record-name. DATATRIEVE inserts these commands into the command file when you use the EXTRACT command. When you invoke the file, DATATRIEVE deletes the incorrect record definition and creates the corrected one.
4. Return to DATATRIEVE.

5. Execute the command file just created by typing the at sign (@) and the file name.

When you name the record, the name must:

- Begin with a letter (A–Z)
- Contain 31 characters or less
- Contain only letters, digits, dollar signs, hyphens, or underscores (A–Z, 0–9, $, -, or _)
- Not duplicate a DATATRIEVE keyword
- Not duplicate the name of an existing dictionary object

5.3 Defining the Parts of a Record Definition

Having named the record, you can define its parts. The complete record definition consists of one or more field definitions for fields like PERSON, EMPLOYEE_STATUS, and EMPLOYEE_NAME in Figure 5–2. Each field definition describes the field itself with a name and a field definition clause. It also describes the field's relationship to other fields with a level number. Follow the definition with a period. Figure 5–3 shows the four parts of the SALARY field in PERSONNEL_REC.

Figure 5–3: Four Parts of the SALARY Field

Level Number	Field Name	
05	SALARY	PIC IS 9(5) EDIT_STRING IS $$$,$$$.

← Two Field Definition Clauses
← Period (.)

ZK-0978A-HC

Following are required parts of the field definition:

- A level number specifying the field's relationship to other fields in the record
- A field name identifying the field
- A period (.) signifying the end of the field definition

In addition, most field definitions contain clauses describing the information stored in the field. Among other things, field definitions can describe the size of a field, the type of information stored in it, and how the information will be displayed.

In Figure 5–3, for example, the PIC (or PICTURE) clause describes the size of the SALARY field and the type of information which can be stored there, in this case a number with no more than five digits. The EDIT_STRING clause specifies that information in the SALARY field will be displayed in monetary format with a leading dollar sign ($) and a comma (,) in the appropriate place.

Level numbers, field names, and field definition clauses are discussed in the following sections.

5.3.1 Specifying Level Numbers

DATATRIEVE recognizes the levels of fields in the record definition according to the level numbers you assign. The level number is the first element of a field definition. Level numbers are 1- or 2-digit numbers, ranging from the highest possible level (1) to the lowest possible level (65). Leading zeros, as in 01 or 05, do not affect the value of the level number.

Figure 5–4 shows the level numbers for fields in the PERSONNEL record definition.

Figure 5–4: Level Numbers in the PERSONNEL Record Definition

```
01 PERSON
    05 ID
    05 EMPLOYEE_STATUS
    05 EMPLOYEE_NAME
        10 FIRST_NAME
        10 LAST_NAME
    05 DEPT
    05 START_DATE
    05 SALARY
    05 SUP_ID
```

The level numbers apply to each group and elementary field:

- Group fields contain one or more group or elementary fields.
- Elementary fields contain one item of data and no other fields.

The group field PERSON is the top-level field and the only field with the level number 01. Every record must have a top-level field. The group field EMPLOYEE_NAME, numbered 05, contains the elementary fields FIRST_NAME and LAST_NAME, numbered 10. The remaining fields, numbered 05 like the group field EMPLOYEE_NAME, are elementary fields at the same level as EMPLOYEE_NAME.

If one of the elementary fields numbered 05 (DEPT, for example), were numbered 06, then that field would no longer be at the same level as the group field preceding it. If DEPT were numbered 06, then it would become a part of the group field EMPLOYEE_NAME.

A group field is not just a marker of record structure and relationships among data items. A group field also gives you a way of using one name to refer to more than one field. You can access all the data in the PERSONNEL_REC record, for example, by using the group field PERSON. The following example shows how you can display all the data in a selected record in the PERSONNEL domain using the PERSON group field in a PRINT statement:

```
DTR> READY PERSONNEL RET
DTR> FIND PERSONNEL RET
[23 records found]
DTR> SELECT 3 RET
DTR> PRINT PERSON RET

                    FIRST       LAST            START              SUP
 ID     STATUS       NAME       NAME    DEPT    DATE      SALARY   ID

02943 EXPERIENCED CASS        TERRY      D98  2-Jan-89    $29,908 39485

DTR>
```

When you develop a record structure, keep in mind the following four guidelines for using group and elementary fields:

- A record definition must define at least one elementary field.
- A record definition with more than one field definition must define a top-level group field that includes all other fields in the record.
- A group field must contain at least one elementary field.
- A group field can contain both elementary and group fields.

Following are rules for level numbers:

- Level numbers need not be consecutive.

  Only the relative value of level numbers determines the relationship between fields. For example, the structure of the record would be no different from its present form if the fields numbered 05 had level numbers 02, and FIRST_NAME and LAST_NAME had level numbers 47. Using similar increments in the numbers of successive levels is convenient but arbitrary. You do not have to use the same increment between levels.

- Only the level numbers determine the relationships among fields.

  The examples of records in this book indent field names to show the relationships among levels of fields. Although indenting fields can make a record definition easy to read, it has no effect on the levels of fields and no effect on the relationships between fields.

- You must use one number for all the fields at the same level in a group field, like FIRST_NAME and LAST_NAME in PERSONNEL_REC.
- The level number for a group field must be lower than the number for any field it contains.

5.3.2 Naming Fields

You must name every field you define. You use field names to control the way DATATRIEVE retrieves, modifies, and stores data. If you do not specify a column header different from the field name, DATATRIEVE uses the field name as the column header when displaying data.

5.3.2.1 Restrictions for Field Names

The names you choose for fields must conform to the general restrictions for DATATRIEVE names, described in the *DATATRIEVE–11 Reference Manual*. In summary, the name:

- Can consist of letters, digits, hyphens, dollar signs, and underscores
- Must begin with a letter
- Must not duplicate a DATATRIEVE keyword
- Must be from 1 to 31 characters long
- Must not duplicate the name of another dictionary object, domain, procedure, or table

In most cases, DATATRIEVE displays an error message if you violate these rules. However, if you duplicate dictionary object names you may not receive an error message, or you may get unexpected results. For example, suppose you want to display the contents of a field with the same name as a readied domain in your workspace. If you enter PRINT and the field name, DATATRIEVE associates the name with the domain rather than the field and displays the contents of the domain. DATATRIEVE associates the name with the field only if it is the second name in a print list.

You can continue a name from one input line to another by typing a hyphen at the end of the input line, pressing RETURN, and completing the name on the next line. To make the original information for the PERSONNEL record (Figure 5–1) conform to the rules for field names, change some of them. None of the names in the original information exceed 30 characters, but several contain spaces, which are illegal characters in DATATRIEVE names. Figure 5–5 shows the necessary changes.

Figure 5–5: Valid Field Names for EMPLOYEE_REC

IDENTIFICATION	→	ID
STATUS	→	EMPLOYEE_STATUS
EMPLOYEE NAME	→	EMPLOYEE_NAME
		FIRST_NAME
		LAST_NAME
DEPARTMENT	→	DEPT
START DATE	→	START_DATE
SUPERVISOR'S IDENTIFICATION NUMBER	→	SUP_ID

ZK-0973A-HC

5.3.2.2 Using Duplicate Field Names

DATATRIEVE does not require field names to be unique. You can have several fields in one record that have the same name. However, fields that share one name must be in different group fields. To refer to a field name that is a duplicate, prefix it with its group field name. The record is a field tree, not a simple linear list. No other field is equivalent to the top-level field. All other fields are at lower levels of the structure. The levels of the field tree define the relationships among group and elementary fields and determine the sequence DATATRIEVE follows in searching for the names of fields. In PERSONNEL_REC the fields ID, EMPLOYEE_STATUS, EMPLOYEE_NAME, DEPT, START_DATE, SALARY, and SUP_ID are at the same level, one level below the top-level PERSON. FIRST_NAME and LAST_NAME are on the third level in the field tree.

The following examples show some consequences of this structure:

- If you specify EMPLOYEE_NAME.LAST_NAME, DATATRIEVE looks at the field names in the group field EMPLOYEE_NAME until it finds the desired field. When DATATRIEVE searches for the specified field, it finds the first field named EMPLOYEE_NAME. Then it looks at the next lower level for the first field named LAST_NAME.

```
DTR> FIND PERSONNEL RET
[23 records found]
DTR> SELECT 3 RET
DTR> PRINT ID, EMPLOYEE_NAME.LAST_NAME, SALARY RET
```

```
          LAST
 ID       NAME     SALARY

02943 TERRY        $29,908

DTR>
```

If you had a duplicate field called LAST_NAME, perhaps under a group field SUP_NAME, you could use the tree structure to tell DATATRIEVE whether you wanted to print EMPLOYEE_NAME.LAST_NAME or SUP_NAME.LAST_NAME.

- LAST_NAME, without any qualification, is also a valid and unique field name in PERSONNEL_REC because there is no duplicate for it in the present record.
- If you tell DATATRIEVE to store a value in ID.FIRST_NAME, it does not find the field FIRST_NAME and does not store the value. It displays the following error message:

```
Field "ID.FIRST_NAME" is undefined or used out of context
```

In general, though you can have duplicate and even multiple field names, it is best to avoid duplicates. Unless you provide a qualified field name when referring to a duplicate field, DATATRIEVE retrieves the value of the first instance of the duplicate field name. If you intend to refer to an instance other than the first but do not specify it, you will receive the output of the first instance instead of what you intend. If you do choose to use duplicate or multiple field names, you should be careful to use qualified names when necessary.

5.3.2.3 Using the Field Name FILLER

Sometimes you want to preserve fields in a data file without using them, usually for one of the following reasons:

- You may not need those fields for a particular application.
- You may want to control the display of records so that you do not display certain data.
- You may want to reserve space in the physical record of the data file.

For these purposes, you can specify the keyword FILLER as the name of an elementary or group field. Like other fields, a field named FILLER must have a level number, and it can contain field definition clauses. Unlike other fields, the field name FILLER can belong to more than one field at the same level in a group field. When you use the PRINT, LIST, MODIFY, STORE, REPORT, and SUM statements, DATATRIEVE ignores values in FILLER

fields. When you use the DISPLAY statement, DATATRIEVE does display the values in FILLER fields.

You cannot retrieve whole records or group fields containing a group field named FILLER. You can, however, retrieve values from the elementary and group fields included in a group field named FILLER. Each of those fields has its own valid name, and you can retrieve the value by specifying that name in a record selection expression, a print list, or a field list.

The following example shows a part of the YACHT record definition with FILLER in place of TYPE, the group field that contains MANUFACTURER and MODEL. It also shows the result of a SHOW FIELDS and two PRINT statements, one of the top-level group field and one of an elementary field (MODEL) in the group field FILLER.

Level numbers and field names of YACHT record with filler as group field:

```
01 BOAT
               03 FILLER
                    06 MANUFACTURER
                    06 MODEL
               03 SPECIFICATIONS
                    06 RIG
                    06 LENGTH_OVER_ALL
                    06 DISPLACEMENT
                    06 BEAM
                    06 PRICE
```

The effect of FILLER on SHOW FIELDS:

```
DTR> SHOW FIELDS RET
YACHTS
   BOAT
      SPECIFICATIONS (SPECS)
         RIG     [Character string]
         LENGTH_OVER_ALL (LOA)   [Character string]
         DISPLACEMENT (DISP)     [Number]
         BEAM    [Number]
         PRICE   [Number]

DTR>
```

The effect of FILLER as a group field on two PRINT statements:

```
DTR> FIND YACHTS; SELECT; PRINT RET

       LENGTH
        OVER
 RIG    ALL    WEIGHT BEAM  PRICE

KETCH   37    20,000  12  $36,951

DTR> PRINT MODEL RET
```

```
  MODEL
37 MK II
DTR>
```

5.3.3 Using Field Definition Clauses

DATATRIEVE handles the information in a field according to the type of data in the field definition. DATATRIEVE recognizes the following field definition clauses:

- COMPUTED BY
- EDIT_STRING
- OCCURS
- PICTURE
- QUERY_HEADER
- QUERY_NAME
- REDEFINES
- SIGN
- USAGE
- VALID IF

When you write a DATATRIEVE record definition, use a PICTURE, USAGE, or COMPUTED BY clause to specify the type of data each elementary field contains.

You can specify the following classes of fields:

- Alphanumeric

  Define an alphanumeric field with a PICTURE clause of the form PIC X(n), where n is an integer value describing the field width. The EMPLOYEE_STATUS field in the record PERSONNEL_REC, for instance, is defined as follows:

  ```
  05 EMPLOYEE_STATUS      PIC IS X(11).
  ```

  You can store any combination of characters in alphanumeric fields: letters, digits, and the special characters that are part of the DATATRIEVE character set. See the *DATATRIEVE–11 Reference Manual* for a definition of the DATATRIEVE character set.

- Numeric

  You can store digits and an optional sign (+ or –) in numeric fields. DATATRIEVE assumes unsigned numbers to be positive in computations. In the YACHT record, DISPLACEMENT, BEAM, and PRICE are numeric fields. Define a numeric field with a PICTURE clause of the form PIC 9(n), where n is an integer value representing the field width, or with any of these USAGE clauses:

  - COMP (or INTEGER)
  - COMP-1 (or REAL)
  - COMP-2 (or DOUBLE)
  - COMP-3 (or PACKED)
  - COMP-5 (or ZONED)

  USAGE clauses follow a field definition, as in the following:

```
06 EMPLOYEE_SALARY      PIC IS 9(6) USAGE IS INTEGER.
```

  See the *DATATRIEVE–11 Reference Manual* for explanations of the internal storage for USAGE clauses.

- DATE

  Define a date field with the USAGE clause in the form USAGE IS DATE. The DATE datatype is an 8-byte field containing date and time values in the form:

```
DD-MMM-YYYY hh:mm:ss.cc
```

  For example, the date and time value of March 2, 1989, 4:29 p.m. is stored as 02-MAR-1989 16:29:00.00. It may also contain values for seconds and hundredths of a second.

  You can store any date and time after 17-Nov-1858 in a date field. You can use the values of date fields in various computations. For example, you can subtract one date from another to get the number of days elapsed between the two dates, or to calculate elapsed time. Refer to the chapter on lexical functions in the *DATATRIEVE–11 Reference Manual* for descriptions of the functions that extract fields from DATE items.

- COMPUTED BY

  Define a computed by field with the COMPUTED BY clause. A computed by field in a record definition does not correspond to a field in the physical record stored in the data file and does not occupy space in a record. It specifies a value expression. For example, you can define a field named SALARY computed by multiplying the hourly pay of an employee (the WAGE field of the same record) by the number of hours worked (the HOURS field of the record):

```
05 SALARY
   EDIT_STRING $$$9.99
   COMPUTED BY WAGE * HOURS.
```

DATATRIEVE computes the value of the field when you refer to it in a statement.

An elementary field can belong to any one of the four classes of fields: alphanumeric, numeric, date, or computed by. You do not need to use a clause to specify the data type for a group field. A group field is always alphanumeric. If you have stored only digits in a group field, you can use it in arithmetic computations.

Table 5–1 summarizes the field classes and their content.

Table 5–1: Field Classes

Field Type	Class	Content
Elementary field	Alphanumeric	Any combination of characters.
	Numeric	Any combination of digits and optional plus (+) or minus (-) sign.
	DATE	A date.
	COMPUTED BY	None; the field definition specifies a value expression, but no value is stored in the record.
Group field	Alphanumeric	The values of the fields contained in the group field.

5.3.4 Specifying Query Names

You can use the QUERY_NAME field definition clause to specify an alternative name for any field in your record definition. When you name a field, it is good to select a field name that is short enough to use easily but long enough to be meaningful, especially to other people who may use the record. At times, however, you may want to use an abbreviation or a memorable short name in place of the formal field name. For example, the YACHT record contains one field named LENGTH_OVER_ALL that is 15 characters long. The name suggests the meaning of the data stored in that field and is therefore helpful to a person unfamiliar with the YACHTS domain. The field has a query name as well, LOA, to save you from typing the complete name each time you refer to the field.

You can use a query name in any DATATRIEVE statement where you can use the corresponding field name. Figure 5–6 shows a set of query names for PERSONNEL_REC.

Figure 5–6: Query Names for PERSONNEL_REC

```
EMPLOYEE_STATUS     QUERY_NAME IS STATUS.
EMPLOYEE_NAME       QUERY_NAME IS NAME.
FIRST_NAME          QUERY_NAME IS F_NAME.
LAST_NAME           QUERY_NAME IS L_NAME.
```

ZK-0972A-HC

5.3.5 Specifying Word Boundary Alignment with the ALLOCATION Clause

The ALLOCATION clause determines which word boundary alignment DATATRIEVE uses when storing records in the data file. You can specify one of three kinds of alignment:

- LEFT_RIGHT
- MAJOR_MINOR
- ALIGNED_MAJOR_MINOR

For explanations of the three word boundary alignments, see the *DATATRIEVE–11 Reference Manual.* If you do not specify otherwise, DATATRIEVE–11 uses LEFT_RIGHT alignment.

If you attempt to use DATATRIEVE–11 on a data file created with an alignment other than LEFT_RIGHT, be sure to specify the matching alignment in the DATATRIEVE–11 record definition. VAX DATATRIEVE, for instance, uses MAJOR_MINOR as the default alignment. You must specify MAJOR_MINOR in your DATATRIEVE–11 record definition if you wish to use a data file created on VAX DATATRIEVE. If the alignment in the record definition does not match the alignment in the data file, you receive the following message:

```
File and domain record lengths don't match (D=57)
```

5.4 Using the OCCURS Clause to Define Hierarchical Records

To save space in your record definition, you may want to define one or more fields as list fields. Compare, for instance, the following two sample output lines from the domain FAMILIES. The first does not use a list field but instead includes a separate field in the record for each child. The second does use a list field:

```
DTR> PRINT FIRST 1 FAMILIES RET

                          KID             KID
  FATHER      MOTHER      NAME     AGE    NAME     AGE

JIM          ANN         URSULA      7   RALPH       3

DTR>

DTR> PRINT FIRST 1 FAMILIES RET

                       NUMBER     KID
  FATHER      MOTHER    KIDS      NAME     AGE

JIM          ANN          2    URSULA       7
                               RALPH        3

DTR>
```

Records without lists are **flat** records, because the elementary fields in them are logically equivalent to each other. When you print a flat record, all the elementary fields display on the same line.

Without an OCCURS clause, the record for FAMILIES can look like the one shown in Figure 5–7.

Figure 5–7: A Flat Record

```
01 FAMILY_REC.
   03 PARENTS.
      06 FATHER PIC X(10).
      06 MOTHER PIC X(10).
   03 KIDS.
      06 FIRST_KID.
         09 KID_NAME PIC X(10).
         09 AGE PIC 99 EDIT_STRING IS Z9.
      06 SECOND_KID.
         09 KID_NAME PIC X(10).
         09 AGE PIC 99 EDIT_STRING IS Z9.
```

When using an OCCURS clause, however, you can define a record with fields that are lists. A record containing a list or lists is not a flat record, but a **hierarchical** record. In hierarchical records, elementary fields are not all logically equivalent to each other. The list field displays on the same line as the elementary fields with the same number, but the list items display on additional lines beneath the list field. With an OCCURS clause for KIDS, the record FAMILY_REC can look like the one in Figure 5–8, a sample record installed with your system.

Figure 5–8: A Hierarchical Record

```
DTR> SHOW FAMILY_REC RET
RECORD FAMILY_REC
01 FAMILY.
   03 PARENTS.
      06 FATHER PIC X(10).
      06 MOTHER PIC X(10).
   03 NUMBER_KIDS PIC 99 EDIT_STRING IS Z9.
   03 KIDS OCCURS 0 TO 10 TIMES DEPENDING ON NUMBER_KIDS.
      06 EACH_KID.
         09 KID_NAME PIC X(10) QUERY_NAME IS KID.
         09 AGE PIC 99 EDIT_STRING IS Z9.
;
DTR>
```

The OCCURS clause in the record definition creates the hierarchical structure. The clause can define a variable-length or a fixed-length list. The following two sections discuss fixed-length and variable-length lists for hierarchical records.

5.4.1 Defining Lists with a Fixed Number of Occurrences

The OCCURS clause format for fixed-length lists is OCCURS n TIMES, where n is the number of occurrences. If you define the record for FAMILIES using OCCURS 2 TIMES, the record definition looks like the following:

```
03 KIDS_NAMES OCCURS 2 TIMES.
   05 FIRST_NAME PIC X(10).
   05 AGE PIC 99 EDIT_STRING IS Z9.
```

The definition specifies that the group field KIDS_NAMES occurs twice. Each occurrence of KIDS_NAMES contains two fields, FIRST_NAME and AGE.

When DATATRIEVE displays the fixed record, the output looks like the following:

```
DTR> PRINT FIRST 2 FAMILIES RET

                          FIRST
                           NAME
  FATHER      MOTHER                    AGE

JIM           ANN         URSULA         7
                          RALPH          3
JIM           LOUISE      ANN           31
                          JIM           29

DTR>
```

You can use OCCURS n TIMES with an elementary or group field, and a record definition can contain any number of OCCURS clauses in this format. That is, an OCCURS clause can contain another fixed-length OCCURS clause.

```
03 KIDS_NAMES OCCURS 2 TIMES
   05 FIRST_NAME PIC X(10).
   05 AGE PIC 99 EDIT_STRING IS Z9.
   05 NICKNAME PIC X(10)
      OCCURS 3 TIMES.
```

If you define a hierarchical record with a list that occurs a fixed number of times, every record in the domain contains enough space to store the same number of list items. If the first two families had only one child, for instance, the FAMILIES domain with a fixed number of occurrences would print a blank line for the second child:

```
DTR> PRINT FAMILIES RET

                          FIRST
                          NAME
  FATHER      MOTHER                  AGE

JIM           ANN         URSULA       7
                                        0
JIM           LOUISE      ANNE        31
                                        0

DTR>
```

A field definition cannot contain both an OCCURS and a COMPUTED BY clause. That is, you cannot specify multiple occurrences of a COMPUTED BY field.

5.4.2 Defining Lists with a Variable Number of Occurrences

Using the OCCURS clause in a field definition defines a hierarchical record that allows a variable number of list items from one record to another. The following format lets you vary the number of list items in the records of a domain:

```
OCCURS min TO max TIMES DEPENDING ON field-name
```

The record definition for FAMILIES uses the OCCURS clause to define KIDS as a variable-length list. The KIDS variable-length list for FAMILY_REC is shown in Figure 5–8. The output of the PRINT command shows the relationship between NUMBER_KIDS and the fields KID_NAME and AGE. The values of KID_NAME and AGE appear as a list in records with more than one kid:

```
DTR> PRINT FAMILIES RET
```

```
                        NUMBER     KID
  FATHER      MOTHER     KIDS      NAME      AGE

JIM           ANN          2    URSULA        7
                                  RALPH         3
JIM           LOUISE       5    ANNE         31
                                  JIM          29
                                  ELLEN        26
                                  DAVID        24
                                  ROBERT       16
JOHN          JULIE        2    ANN          29
                                  JEAN         26
JOHN          ELLEN        1    CHRISTOPHR    0
ARNIE         ANNE         2    SCOTT         2
                                  BRIAN         0
SHEARMAN      SARAH        1    DAVID         0
TOM           ANNE         2    PATRICK       4
                                  SUZIE         6
BASIL         MERIDETH     6    BEAU         28
                                  BROOKS       26
                                  ROBIN        24
                                  JAY          22
                                  WREN         17
                                  JILL         20
ROB           DIDI         0
JEROME        RUTH         4    ERIC         32
                                  CISSY        24
                                  NANCY        22
                                  MICHAEL      20
TOM           BETTY        2    MARTHA       30
                                  TOM          27
GEORGE        LOIS         3    JEFF         23
                                  FRED         26
                                  LAURA        21
HAROLD        SARAH        3    CHARLIE      31
                                  HAROLD       35
                                  SARAH        27
EDWIN         TRINITA      2    ERIC         16
                                  SCOTT        11

DTR>
```

When you define a record with a variable-length list in it, you must put the list at the end of the record definition. You can use only one field with an OCCURS clause. That is, you cannot have an OCCURS clause within an OCCURS clause. If you attempt to define another field after an OCCURS clause and at the same level, you receive the following error message:

```
ONLY Subordinate fields allowed after OCCURS DEPENDING ON
```

5.4.3 Nesting Lists Within Lists to Form Sublists

Although you can use only one OCCURS clause in a record definition, you can define fixed-length lists within a variable-length list. In fact, you can include any number of OCCURS n TIMES clauses within a field defined with an OCCURS clause. This sample record definition with a sublist is an extension of the FAMILY record. The list PET occurs twice for each kid, so each kid in each family can record the data for two pets:

```
DTR> SHOW PETS RET
DOMAIN PETS
USING PET_REC ON PET.DAT;

DTR> SHOW PET_REC RET
RECORD PET_REC
01 FAMILY.
   03 PARENTS.
      06 FATHER PIC X(10).
      06 MOTHER PIC X(10).
   03 NUMBER_KIDS PIC 99 EDIT_STRING IS Z9.
   03 KIDS OCCURS 0 TO 10 TIMES DEPENDING ON NUMBER_KIDS.
      06 EACH_KID.
         09 KID_NAME PIC X(10) QUERY_NAME IS KID.
         09 KID_AGE PIC 99 EDIT_STRING IS Z9.
         09 PET OCCURS 2 TIMES.
            13  PET_NAME  PIC X(10).
            13  PET_AGE   PIC 99.
;

DTR> READY PETS RET
DTR> PRINT FIRST 2 PETS RET
```

FATHER	MOTHER	NUMBER KIDS	KID NAME	KID AGE	PET NAME	PET AGE
JIM	LORAINE	2	GARY	24	POP	03
					SODA	04
			SUE	23	MOUSE	03
					SHORTY	08
JIM	ANN	2	URSULA	7	SQUEEKY	03
					FRANK	07
			RALPH	3		00
						00

5.4.4 Changing the Length of a List

If you define a record with the OCCURS clause, you can change the number of occurrences in a list. If you specify the MAX clause when defining the data file, all records have enough space for the maximum number of occurrences, and you can always change the number of occurrences up to the limit you have set:

```
DTR> DEFINE FILE FOR FAMILIES MAX
```

If you do not specify the MAX clause when you define a sequential file, DATATRIEVE sets the length of each record when you store it. In that case, you cannot add examples to the list. You can always decrease the number of occurrences, though, and you can increase the number of occurrences back to the original number.

If you do not specify the MAX clause and the file organization is indexed, you can still change the number of occurrences.

The following example shows how to add items to a list. Because you are modifying an existing record, you use MODIFY rather than STORE to add items to the list:

```
DTR> READY INDEXED_FAMILIES WRITE RET
DTR> FIND FIRST 1 INDEXED_FAMILIES RET
[1 Record found]
DTR> SELECT; PRINT RET

                        NUMBER     KID
  FATHER      MOTHER     KIDS      NAME     AGE

JIM         ANN            2    URSULA       7
                                 RALPH        3

DTR> MODIFY NUMBER_KIDS RET
Enter NUMBER_KIDS: 4 RET
DTR> FIND KIDS RET
[4 records found]
DTR> SELECT 3 RET
DTR> MODIFY RET
Enter KID_NAME: NICKY RET
Enter AGE: 2 RET
DTR> SELECT 4 RET
DTR> MODIFY RET
Enter KID_NAME: TAM RET
Enter AGE: 1 RET
DTR> PRINT FIRST 1 INDEXED_FAMILIES RET

                        NUMBER     KID
  FATHER      MOTHER     KIDS      NAME     AGE

JIM         ANN            4    URSULA       7
                                 RALPH        3
                                 NICKY        2
                                 TAM          1
DTR>
```

For information on retrieving data from hierarchical records, see Chapter 14 in this manual.

Chapter 6

Defining Files

The way you define a data file determines how much storage space the file occupies, how quickly you can retrieve data from the file, and whether you can change or duplicate data fields in that file.

DATATRIEVE is based on RMS, the standard DIGITAL record and file management software facility. Based on how you define your data files, DATATRIEVE uses RMS to create, define, store, manipulate, and maintain information within your files. Your operating system documentation will give you more information on RMS.

You can define two types of files in DATATRIEVE:

- Sequential files, which store records in the order you enter them
- Indexed files, which store records according to the order of a specified key field

This chapter discusses the DEFINE FILE command, the choices you have when deciding whether to use sequential or indexed files, and other options available to you when you define a file in DATATRIEVE.

6.1 Choosing a Sequential or an Indexed File

Sequential files require less storage space than indexed files, but it often takes longer to retrieve data from sequential files.

To retrieve records from a sequential file, DATATRIEVE searches records one by one according to their order in the file. To retrieve records from an indexed file, DATATRIEVE searches the file according to specified key fields. The key fields are searched first, regardless of their order in the data file.

Sequential organization is useful in certain applications. For example, if your records contain a date field and you frequently retrieve them in chronological order, it is often best to arrange the records in the order you stored them. You are likely to want sequential access to banking transactions, for example.

Use sequential files for accessing large groups of records in the order you stored them.

In other cases, sequential organization may be unnecessarily slow. When you use a record selection expression (RSE) to form a record stream or collection of records from a sequential file, DATATRIEVE has to start at the beginning of the file and read every record until it finds the ones that you request.

If you use an RSE to form a record stream based on key fields, RMS searches through the index it maintains without having to read every record. If you need to access a small number of records distributed throughout a file, use an indexed file.

6.1.1 Modifying and Deleting Records

Because of differences in file organization, you modify and delete records differently for sequential and indexed files. In a sequential file, you can use the MODIFY statement to change any field, but in an indexed file, you cannot modify the primary key field. You also cannot modify secondary key fields defined with the NO CHANGE clause explained in the section on optional clauses later in this chapter.

RMS does not allow you to use the ERASE statement in a sequential file where sequence is the basis of the file organization. If you need to erase records from your data file, use an indexed file. You can, however, use the MODIFY statement on records in sequential files and change every field to zero or spaces depending on the data type.

6.1.2 Summary of Differences

Table 6–1 summarizes the differences between sequential and indexed files.

Table 6–1: A Comparison of Sequential and Indexed Files

Sequential File	Indexed File
Stores records in the order you can create them	Stores records according to the values in the primary key field
Takes up less storage space than an indexed file	Allows you to retrieve certain data faster than a sequential file does
Allows you to modify any field	Does not allow you to modify primary key field or any key field that has NO CHANGE attribute
Requires you to use MODIFY instead of ERASE	Lets you use ERASE statement

6.2 Defining a File Using the DEFINE FILE Command

DATATRIEVE can store and retrieve data using existing RMS data files, so it is compatible with other languages or utilities that use RMS. It can also create RMS files, including sequential files and multikey indexed files. The DEFINE FILE command forms an RMS data file for the domain you specify. It uses the following format:

```
DEFINE FILE [ FOR ] domain-name [,]

    [ ALLOCATION = n    ]
    [ SUPERSEDE         ] [, . . . ]
    [ MAX               ]
  { KEY = field-name-1 [([NO] CHANGE[,] [NO] DUP)] } [, . . . ]
```

The *DATATRIEVE–11 Reference Manual* also discusses the DEFINE FILE command.

6.2.1 Defining a Sequential File

If you decide you want your data file to be sequential, follow these steps:

1. Before you define a data file, the associated domain and record definitions must be in your dictionary. You can use the SHOW command to be sure that they are in place:

```
DTR> SHOW YACHTS_SEQUENTIAL, YACHT RET
DOMAIN YACHTS_SEQUENTIAL
 USING YACHT ON YACHT.SEQ ;
RECORD YACHT
USING
01 BOAT.
  03 TYPE.
     06 MANUFACTURER PIC X(10)
        QUERY_NAME IS BUILDER.
     06 MODEL PIC X(10).
  03 SPECIFICATIONS
     QUERY_NAME SPECS.
     06 RIG PIC X(6)
        VALID IF RIG EQ "SLOOP","KETCH","MS","YAWL".
     06 LENGTH_OVER_ALL PIC XXX
        VALID IF LOA BETWEEN 15 AND 50
        QUERY_NAME IS LOA.
     06 DISPLACEMENT PIC 99999
        QUERY_HEADER IS "WEIGHT"
        EDIT_STRING IS ZZ,ZZ9
        QUERY_NAME IS DISP.
     06 BEAM PIC 99.
     06 PRICE PIC 99999
        VALID IF PRICE>DISP*1.3 OR PRICE EQ 0
        EDIT_STRING IS $$$,$$$.
;
DTR>
```

2. Use the DEFINE FILE command to define the file. For a sequential file, you need to specify only the domain name in the DEFINE FILE command:

```
DTR> DEFINE FILE FOR YACHTS_SEQ RET
DTR>
```

3. Select the options you want to use with the file, as explained in Section 6.2.3

6.2.2 Defining an Indexed File

You use the KEY clause in the DEFINE FILE command to create an indexed file. The clause creates an indexed file and specifies a field in the record definition to be an index key for the domain's data file. The first key field you name in the DEFINE FILE command is the primary key. All subsequent keys are alternate keys.

If you decide you want your file to be indexed, first analyze your record definition to decide which field or fields you want to be key fields.

You can designate only one primary key field, but you can name as many alternate key fields as you wish from the remaining fields in the record. DATATRIEVE searches the primary and alternate fields independently, so defining alternate keys does not slow performance for queries based on the primary key.

To choose a primary key, decide which field of the record you are likely to name most often in queries. Make certain you will not want to change that field, because DATATRIEVE does not allow you to change the values in primary keys.

Finally, look for a field that has a unique value for each record. Unless you specify otherwise, DATATRIEVE does not allow you to have the same value in more than one primary key field. For instance, you could not have two records in PERSONNEL with the ID 99883. You can use the DUP option to allow duplicates, as explained later in this chapter, but duplicate values slow performance.

If you were setting up a PERSONNEL domain, you might predict that most users seeking information on an employee would base their search on the ID. You would not want to change identification numbers, and no two employees should have the same identification number. Consequently, ID uniquely identifies a record and is a good primary key for your PERSONNEL domain. This DEFINE FILE command would make ID the primary key for the indexed file PERSONNEL:

```
DTR> DEFINE FILE FOR PERSONNEL, KEY=ID RET
DTR>
```

6.2.2.1 Using a Group Field as the Primary Key

If the field you use most often does not uniquely identify each record, you can find another field that, together with the first, does identify the record. Then designate a group field made up of the two fields as the primary key. In the domain YACHTS the elementary field MANUFACTURER does not uniquely identify a record. ALBIN, for instance, is the builder of three of the first five YACHTS:

```
DTR> PRINT FIRST 5 YACHTS RET
```

```
                                LENGTH
                                 OVER
MANUFACTURER    MODEL      RIG    ALL    WEIGHT BEAM  PRICE

 ALBERG       37 MK II   KETCH   37     20,000  12  $36,951
 ALBIN        79         SLOOP   26      4,200  10  $17,900
 ALBIN        BALLAD     SLOOP   30      7,276  10  $27,500
 ALBIN        VEGA       SLOOP   27      5,070  08  $18,600
 AMERICAN     26         SLOOP   26      4,000  08   $9,895

DTR>
```

The combination of MANUFACTURER and MODEL, however, is unique for each record. These two fields are defined as the group field TYPE in the YACHT record definition. Therefore, TYPE, which has no duplicate values, makes a suitable primary key.

Note that when you use a group field as a primary key, you cannot modify any of the elementary fields in the group. In addition, note that BUILDER has been defined as a query name for MANUFACTURER in the YACHT record definition. This means you can use the shorter name BUILDER in place of MANUFACTURER in queries.

If a group field is the primary key, list the field most commonly used in queries as the first elementary field in the group field. The field MANUFACTURER (BUILDER) appears in queries more frequently than MODEL, for example.

When a group field such as TYPE is defined as a key field, keyed access will work only for queries involving the group field itself or the first elementary field in the group. With TYPE as the key field, DATATRIEVE conducts an indexed search for TYPE or BUILDER but a sequential search for the second elementary field, MODEL.

If you want DATATRIEVE to use the first elementary field in a group field as a key, you must define that first field as either PIC X or PIC 9 in the group field definition. When you ready the domain, DATATRIEVE uses the group field as the key. If the first elementary field is anything but a simple numeric or character string, it is not treated as a key when the domain is readied.

6.2.2.2 Defining Alternate Keys

If there are additional fields that you often use in queries, you can define them as alternate keys. DATATRIEVE performs an indexed search when you refer to an alternate key in a query.

For example, you could make LENGTH_OVER_ALL an alternate key for YACHTS, using the following DEFINE FILE command:

```
DTR> DEFINE FILE FOR YACHTS KEY = TYPE, KEY = LOA RET
DTR>
```

DATATRIEVE allows duplicate values for the alternate keys unless you specify otherwise with the NO DUP option explained in Section 6.2.3.

Chapter 17 in this manual describes how to use key fields for optimum processing.

6.2.2.3 Summary of Rules for Defining Key Fields

To set up key fields that DATATRIEVE can process most efficiently, use the following guidelines:

- When defining data, make the field most commonly used in queries the primary key.
- If the most commonly used field does not uniquely identify a record, combine it with another field in a field group so that the group field identifies the record.
- Avoid duplicate values of a primary key when possible because duplications slow performance.
- If you decide to make a group field the primary key, list the field most commonly used in queries as the first elementary field.
- If your record has other fields you often use with the primary key in queries, designate them as alternate keys.

6.2.3 Optional Clauses with the DEFINE FILE Command

You can use the following options in your DEFINE FILE command for both sequential and indexed files:

- ALLOCATION = n

  ALLOCATION in the DEFINE FILE command refers to the allocation of disk blocks to a file. If you do not specify an allocation for a sequential file that you create, DATATRIEVE allots 0 blocks to the file and then assigns space as needed. For an indexed file, it allots a small space when you create the file and additional space as needed.

  If you know your file is going to be large, however, you can use the ALLOCATION = n clause to reserve storage space for the file and increase the speed at which you can store records:

```
DTR> DEFINE FILE FOR YACHTS ALLOCATION = 2000 RET
DTR>
```

See Chapter 17 for a discussion of techniques for optimizing response time and workspace.

- SUPERSEDE

  If you specify SUPERSEDE in your DEFINE FILE command, DATATRIEVE deletes the existing file with the same name and replaces it with the new one.

  Because RSTS/E systems keep only one version of a file, you must specify SUPERSEDE on RSTS/E systems when defining a file to replace one that already exists. If you do not, DATATRIEVE sends you an error message and does not create the new file:

```
DTR> DEFINE FILE FOR YACHTS RET
File "YACHT.DAT" already exists
DTR>
```

  On RSX systems, be sure that the complete file specification, including version number, duplicates the one you want to replace. Otherwise, DATATRIEVE keeps both the old and the new files. If the file you want to replace is YACHT.DAT;1, you would specify the replacement file as follows:

```
DTR> DEFINE DOMAIN YACHTS USING YACHT ON YACHT.DAT;1 RET
DTR> DEFINE FILE FOR YACHTS SUPERSEDE RET
DTR>
```

- MAX

  As explained in Chapter 5, a record defined with the OCCURS clause allows you to vary the number of occurrences in a list.

  For a record with an OCCURS clause, use the MAX clause when defining the file. This reserves space in every record for the maximum possible number of list items. The record for FAMILIES, for instance, has the following OCCURS clause:

```
03 KIDS OCCURS 0 TO 10 TIMES DEPENDING ON NUMBER_KIDS.
```

  If you request the MAX option when defining a file for FAMILIES, you reserve space for 10 kids in each record. Use the following form of the command:

```
DTR> DEFINE FILE FOR FAMILIES MAX RET
DTR>
```

  If you do not specify the MAX clause when you define a sequential file, DATATRIEVE sets the length of each record when you store it. In that case, you cannot add additional elements to the list. You can decrease the number of occurrences, and you can also increase the number of occurrences back to the original number.

If you specify more than one option in the DEFINE FILE command, separate each from the next with a comma. You do not have to specify the options in any particular order.

```
DTR> DEFINE FILE FOR FAMILIES SUPERSEDE, ALLOCATION = 2000, MAX RET
DTR>
```

You can use the following optional clauses with indexed files:

- CHANGE or NO CHANGE

  The CHANGE or NO CHANGE clause determines whether or not you can modify the contents of the key field associated with the clause. To specify the CHANGE or NO CHANGE option, put the clause in parentheses after the name of the key field:

  ```
  DTR> DEFINE FILE FOR YACHTS KEY = TYPE (NO CHANGE) RET
  DTR>
  ```

  You cannot specify CHANGE for a primary key. CHANGE is in effect for alternate keys unless you specify otherwise.

- DUP or NO DUP

  The DUP or NO DUP clause determines whether or not you can assign the same value to the specified key field in more than one record. Can more than one YACHTS record, for instance, have the value ALBIN for the key field BUILDER?

  NO DUP is the default for primary keys. However, DATATRIEVE allows you to specify DUP for primary keys. You may slow performance in retrieving records if you do so, because DATATRIEVE performs an indexed search to find the primary key and then a sequential search through the duplicates when it finds them.

  For alternate keys you can, by default, use duplicate values. You can specify NO DUP if you like, but eliminating duplicates from an alternate key field can limit the number of records you can store successfully. If you assigned RIG as an alternate key and specified the NO DUP option, for instance, you could store only four records in YACHTS: one sloop, one ketch, one yawl, and one MS.

  If you do not want duplicates and you do not want the alternate key values to change, put both sets of keywords in parentheses and separate them with a comma: KEY = field-name (NO DUP, NO CHANGE).

The following example defines an indexed file for YACHTS. It uses the group field TYPE as the primary key with the DUP option in effect, and RIG as an alternate key with the NO CHANGE option in effect:

```
DTR> DEFINE FILE FOR YACHTS KEY = TYPE (DUP), RET
DFN>      KEY = RIG (NO CHANGE) RET
DTR>
```

Chapter 7

Limiting Record Streams with Record Selection Expressions

You define and store data so you can retrieve information in whatever form is most useful. You may want to perform any of the following activities:

- Display a group of records (PRINT or REPORT statements)
- Form a temporary collection of records (FIND statement)
- Update a group of records (MODIFY statement)

To carry out any of these tasks, you must identify a **record stream**, that is, a group of records from a domain or collection. You form record streams with DATATRIEVE by specifying a *record selection expression* (RSE).

By including various clauses in the RSE, you can determine the content of the record stream in several ways:

- By specifying the number of records in the record stream (FIRST n clause)
- By limiting the record stream to records that meet a conditional test (WITH clause)
- By sorting the records according to the values of one or more fields (SORTED BY clause).

This chapter presents many examples to teach you how to use RSEs. The examples use RSEs with the PRINT statement, but you may use them with FIND, REPORT, or other DATATRIEVE statements.

In addition, see Chapter 14 for information about another form of RSE that lets you access list items from hierarchical records.

7.1 Accessing All the Records in a Domain

If a domain does not contain many records, you may be satisfied to display all of the records. You form one type of PRINT statement by typing PRINT followed by an RSE; for example:

```
DTR> READY YACHTS RET
DTR> PRINT YACHTS RET

                                  LENGTH
                                   OVER
MANUFACTURER    MODEL      RIG     ALL    WEIGHT BEAM  PRICE

 ALBERG         37 MK II   KETCH   37     20,000  12   $36,951
 ALBIN          79         SLOOP   26      4,200  10   $17,900
 ALBIN          BALLAD     SLOOP   30      7,276  10   $27,500
 ALBIN          VEGA       SLOOP   27      5,070  08   $18,600
 AMERICAN       26         SLOOP   26      4,000  08    $9,895
 AMERICAN       26-MS      MS      26      5,500  08   $18,895
 BAYFIELD       30/32      SLOOP   32      9,500  10   $32,875
 BLOCK I.       40         SLOOP   39     18,500  12
 BOMBAY         CLIPPER    SLOOP   31      9,400  11   $23,950
 BUCCANEER      270        SLOOP   27      5,000  08
 BUCCANEER      320        SLOOP   32     12,500  10
 C&C            CORVETTE   SLOOP   31      8,650  09

   .              .          .      .        .     .      .
   .              .          .      .        .     .      .
   .              .          .      .        .     .      .

 VENTURE        222        SLOOP   22      2,000  07    $3,564
 WESTERLY       CENTAUR    SLOOP   26      6,700  08   $15,245
 WESTSAIL       32         SLOOP   32     19,500  11
 WINDPOWER      IMPULSE    SLOOP   16        650  07    $3,500
 WRIGHT         SEAWIND II SLOOP   32     14,900  00   $34,480

DTR>
```

The PRINT YACHTS statement gives a display of all the records in the YACHTS domain. The source for the RSE is YACHTS, the name of the domain. Each RSE must include a source for the records, either a domain name, collection name, or list name.

For clarity, you may want to specify the keyword ALL when you want a record stream to include all the records in a domain. The keyword ALL is optional. For example, PRINT ALL YACHTS and PRINT YACHTS are equivalent.

7.2 Specifying the Number of Records in the Record Stream

The keywords ALL and FIRST let you indicate the number of records in the record stream. To specify the number of records in the record stream, type FIRST followed by a number before typing the source for the RSE. For example:

```
DTR> PRINT FIRST 5 YACHTS RET

                                LENGTH
                                 OVER
MANUFACTURER     MODEL     RIG   ALL   WEIGHT BEAM  PRICE

 ALBERG        37 MK II   KETCH  37    20,000  12  $36,951
 ALBIN         79         SLOOP  26     4,200  10  $17,900
 ALBIN         BALLAD     SLOOP  30     7,276  10  $27,500
 ALBIN         VEGA       SLOOP  27     5,070  08  $18,600
 AMERICAN      26         SLOOP  26     4,000  08   $9,895

DTR>
```

In this case FIRST 5 YACHTS is the RSE. DATATRIEVE displays the first five records in YACHTS according to their order in the data file. An RSE can have either of the following forms, where n is any number less than or equal to the total number of records in the domain or collection:

FIRST n domain-name ... for a domain
FIRST n collection-name ... for a collection

If n is greater than the number of records in the source, DATATRIEVE gives you all the records that fulfill the RSE and does not display a message on your terminal.

Limiting the record stream can be useful when you are testing procedures, complex RSEs, or report specifications. You can conduct your tests without having to wait for DATATRIEVE to display the complete set of records.

Specifying the number of records can be useful, too, when you want to display a fixed number of those records that meet the requirements of the RSE:

```
DTR> PRINT FIRST 5 YACHTS WITH PRICE NE 0 RET

                                LENGTH
                                 OVER
MANUFACTURER     MODEL     RIG   ALL   WEIGHT BEAM  PRICE

 ALBERG        37 MK II   KETCH  37    20,000  12  $36,951
 ALBIN         79         SLOOP  26     4,200  10  $17,900
 ALBIN         BALLAD     SLOOP  30     7,276  10  $27,500
 ALBIN         VEGA       SLOOP  27     5,070  08  $18,600
 AMERICAN      26         SLOOP  26     4,000  08   $9,895

DTR>
```

7.3 Identifying Records with Conditional Expressions

There are several ways to limit the number of records in the record stream. Often you are interested in grouping similar records together, regardless of their position in the domain or record stream. You can restrict the record stream to those records that satisfy a specified condition by using the WITH clause of the RSE. Different forms of the WITH clause specify different types of relationships between the values of the same field for different records. You can form record streams based on:

- Patterns among the field values (EQUAL, NOT EQUAL, CONTAINING)
- Field values that fall within a specified range (BETWEEN . . . AND . . . , LESS THAN, GREATER THAN)
- Field values you can or cannot find in a table

7.3.1 Comparing Records by Pattern Recognition

You can group records if the characters of a field value are equal or not equal to a specified value; for example:

```
DTR> PRINT YACHTS WITH BUILDER = "ALBIN" RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL    WEIGHT BEAM  PRICE

 ALBIN          79        SLOOP  26      4,200  10  $17,900
 ALBIN          BALLAD    SLOOP  30      7,276  10  $27,500
 ALBIN          VEGA      SLOOP  27      5,070  08  $18,600

DTR>
```

This statement asks DATATRIEVE to examine each record of the YACHTS domain and display only those records with the value "ALBIN" for the BUILDER field. After testing each record of YACHTS, DATATRIEVE identifies and then displays each record that meets the specified condition. WITH BUILDER = "ALBIN" lets you limit the record stream to the records you wish to access.

The expression, BUILDER = "ALBIN", is a **Boolean expression**. A Boolean expression controls a comparison between value expressions. A Boolean expression is either true or false depending on the values of the field and the value expression specified. The term that relates the value expressions is called a **relational operator**. In this example the relational operator is the equal sign (=).

When you use EQUAL (=) or NOT EQUAL, you can list more than one value expression in the same Boolean expression. The following queries specify a group of value expressions for DATATRIEVE to compare with each field value. The comma here is equivalent to saying AND BUILDER =, so that the statement tells DATATRIEVE to print all yachts by ALBIN and all yachts by ALBERG:

```
DTR> PRINT YACHTS WITH BUILDER = "ALBIN", "ALBERG"

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL    WEIGHT BEAM  PRICE

 ALBIN          79        SLOOP  26      4,200  10  $17,900
 ALBIN          BALLAD    SLOOP  30      7,276  10  $27,500
 ALBIN          VEGA      SLOOP  27      5,070  08  $18,600
 ALBERG         37 MK II  KETCH  37     20,000  12  $36,951

DTR> PRINT YACHTS WITH RIG NOT EQUAL "SLOOP", "KETCH"

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL    WEIGHT BEAM  PRICE

 AMERICAN       26-MS     MS     26      5,500  08  $18,895
 EASTWARD       HO        MS     24      7,000  09  $15,900
 FJORD          MS 33     MS     33     14,000  11
 LINDSEY        39        MS     39     14,500  12  $35,900
 ROGGER FD      M/S       MS     35     17,600  11

DTR>
```

Note that the EQUAL (=) and NOT EQUAL operators are case sensitive. They see uppercase and lowercase letters as different:

```
DTR> FIND YACHTS WITH BUILDER = "Albin" RET
[0 records found]
DTR> FIND YACHTS WITH BUILDER = "ALBIN" RET
[3 records found]
```

Because the builders' names are in uppercase letters in the data file but lowercase letters in the first query, DATATRIEVE did not find any record for a builder named "Albin". However, for "ALBIN", it found three records.

On the other hand, the CONTAINING operator is indifferent to the case of the letters. It finds matches if there is agreement with all of the letters in the field value or with a substring derived from the field value. Thus the CONT operator finds the "ALBIN" record if you specify either "Albin" or "bin", a three letter substring:

```
DTR> FIND YACHTS WITH BUILDER CONT "Albin" RET
[3 records found]
DTR> PRINT YACHTS WITH BUILDER CONT "bin" RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL    WEIGHT BEAM  PRICE

 ALBIN          79        SLOOP  26      4,200  10   $17,900
 ALBIN          BALLAD    SLOOP  30      7,276  10   $27,500
 ALBIN          VEGA      SLOOP  27      5,070  08   $18,600

DTR>
```

DATATRIEVE finds and displays each record that contains the substring "bin" in the value for BUILDER.

Note that another difference between EQUALS and CONTAINING is that DATATRIEVE can optimize EQUALS if the field is an RMS key, but it cannot optimize for CONTAINING. The CONTAINING operator always reads every record in the file. The EQUALS operator does not have to read each record if the field is a key.

7.3.2 Grouping Records for a Range of Values

DATATRIEVE allows you to use a variety of relational operators to test whether a field value for a record falls within a specified range. These operators are GREATER_THAN (> or GT), GREATER_EQUAL (GE), LESS_THAN (< or LT), LESS_EQUAL (LE), and BETWEEN (BT):

```
DTR> PRINT YACHTS WITH PRICE GREATER_THAN 50000 RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL    WEIGHT BEAM  PRICE

 CHALLENGER     41        KETCH  41     26,700  13   $51,228
 ISLANDER       FREEPORT  KETCH  41     22,000  13   $54,970
 OLYMPIC        ADVENTURE KETCH  42     24,250  13   $80,500

DTR> PRINT YACHTS WITH PRICE GREATER_EQUAL 50000 RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL    WEIGHT BEAM  PRICE

 CHALLENGER     41        KETCH  41     26,700  13   $51,228
 ISLANDER       FREEPORT  KETCH  41     22,000  13   $54,970
 NORTHERN       37        KETCH  37     14,000  11   $50,000
 OLYMPIC        ADVENTURE KETCH  42     24,250  13   $80,500

DTR>
```

Note the difference between the two record streams. NORTHERN, priced at exactly $50,000, appears when the Boolean expression is PRICE GREATER_EQUAL 50000, but it does not appear when you use the GREATER_THAN operator.

The LESS_THAN and LESS_EQUAL operators work in a similar manner. The LESS_EQUAL operator includes a record if its field is either *less than or equal to* the value expression specified.

The BETWEEN operator is the equivalent of the GREATER_EQUAL and LESS_EQUAL operators combined. It searches for records with field values that are within the range specified or equal to either of the value expressions that determine the range. For the BETWEEN operator to work, the range must go from a smaller value to a larger one. In the following example, the Boolean expression identifies a record stream that includes records with values for PRICE between $50,000 and $90,000:

```
DTR> PRINT YACHTS WITH PRICE BETWEEN 50000 AND 90000 RET

                                    LENGTH
                                     OVER
MANUFACTURER    MODEL       RIG      ALL    WEIGHT BEAM  PRICE

 CHALLENGER  41          KETCH    41     26,700  13  $51,228
 ISLANDER    FREEPORT    KETCH    41     22,000  13  $54,970
 NORTHERN    37          KETCH    37     14,000  11  $50,000
 OLYMPIC     ADVENTURE   KETCH    42     24,250  13  $80,500

DTR>
```

7.3.3 Grouping Records by Reference to a Table

Some domains are associated with dictionary tables containing code strings that correspond to values in a field in the record. You can form an RSE that causes DATATRIEVE to look up the field value in the table. You can use the relational operator IN to compare the contents of a field with the code strings in a dictionary table or domain table. If there is a match on the code string in the table, DATATRIEVE includes the record in the record stream. Two queries using table-based RSEs are FIND YACHTS WITH RIG IN RIG_TABLE and FIND YACHTS WITH RIG NOT IN RIG_TABLE.

See Chapter 12 for a discussion of tables.

7.3.4 Summary of the Relational Operators

Table 7–1 summarizes all of the relational operators available to form Boolean expressions in the WITH clause of an RSE.

Table 7–1: Conditional Comparisons for an RSE

Type of Comparison	Relationship of Values in Boolean	Relational Operator	Boolean Expression
Pattern recognition	Exact match (case sensitive)	= EQUAL EQ	BUILDER = "ALBIN" "ALBIN" = BUILDER
	No match (case sensitive)	NE NOT_EQUAL NOTEQUAL	BUILDER NE "ALBIN" "ALBIN" NE BUILDER
	Substring matches (not case sensitive)	CONT CONTAINING	BUILDER CONT "bin"
	Substring does not match (not case sensitive)	NOT CONT NOT CONTAINING	BUILDER NOT CONT "bin"
Value within a range	Value is greater than	> GT GREATER_THAN	PRICE > 50000 50000 > PRICE
	Value is greater than or =	GE GREATER_EQUAL	PRICE GE 50000 50000 GE PRICE
	Value is less than	< LT LESS_THAN	PRICE < 20000 20000 < PRICE
	Value is less than or =	LE LESS_EQUAL	PRICE LE 20000
	Value is between the two values or = to one	BT BETWEEN	PRICE BETWEEN 30000 AND 54000
Look up in table	Field value is in the table	IN table-name	RIG IN RIG_TABLE
	Field value is not in the table	NOT IN table-name	RIG NOT IN RIG_TABLE

(continued on next page

Table 7–1 (Cont.): Conditional Comparisons for an RSE

Type of Comparison	Relationship of Values in Boolean	Relational Operator	Boolean Expression
Record stream empty	Record stream is not empty	ANY rse	FAMILIES WITH ANY KIDS
	Record stream is empty	NOT ANY rse	FAMILIES WITH NOT ANY KIDS

7.3.5 Setting Up Multiple Tests with Compound Booleans

Thus far, each Boolean expression imposed just one test for records to be included in the record stream. To set up multiple or complex tests for records, you can join two or more Boolean expressions together. Expressions that join Booleans are called **Boolean operators**.

There are four Boolean operators: AND, OR, NOT, and BUT. With AND, OR, and BUT you can join two or more Boolean expressions together to form a single Boolean expression. NOT allows you to reverse the value of a Boolean expression.

If you link Boolean expressions with AND or BUT, the resulting Boolean expression is true only if all the Booleans linked with AND or BUT are true.

If you link Boolean expressions with OR, the resulting Boolean expression is true if any one of the Booleans linked with OR is true.

If you precede a Boolean expression with NOT, the resulting Boolean expression is true if the Boolean expression following NOT is false.

The following example shows the use of the Boolean operator:

```
DTR> PRINT YACHTS WITH RIG = MS OR LOA = 39 RET

                                  LENGTH
                                   OVER
MANUFACTURER    MODEL      RIG     ALL    WEIGHT BEAM  PRICE

 AMERICAN      26-MS       MS       26      5,500  08  $18,895
 BLOCK I.      40          SLOOP    39     18,500  12
 EASTWARD      HO          MS       24      7,000  09  $15,900
 FJORD         MS 33       MS       33     14,000  11
 LINDSEY       39          MS       39     14,500  12  $35,900
 PEARSON       39          SLOOP    39     17,000  12
 ROGGER FD     M/S         MS       35     17,600  11

DTR>
```

The query displays data on all yachts that have a RIG that is MS or an LOA equal to 39. For DATATRIEVE to include a record in the record stream, it must find that the record from YACHTS satisfies either condition or both.

7.4 Sorting the Record Stream by Field Values

When you use a PRINT statement to display a record stream, the primary key defined for the data file determines the order of the records. However, you can use the SORTED BY clause of the RSE to sort the record stream in a different order. For example, the records in YACHTS are already sorted by BUILDER, the first part of the primary key (TYPE) for the data file.

If you are interested in the length of the boats, you can sort the records by LOA. To break down each length yacht by weight, specify DISP, the query name for displacement, as an additional sort key. The following query first sorts the YACHTS records according to LOA and DISP, then limits the record stream to the first five records:

```
DTR> FIND YACHTS SORTED BY LOA, DISP RET
[113 records found]
DTR> PRINT FIRST 5 CURRENT RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 WINDPOWER    IMPULSE    SLOOP   16      650  07   $3,500
 CAPE DORY    TYPHOON    SLOOP   19    1,900  06   $4,295
 ENCHILADA    20         SLOOP   20    2,300  07
 SAN JUAN     21         SLOOP   21    1,250  07
 VENTURE      21         SLOOP   21    1,500  07   $2,823

DTR>
```

The SORTED BY clause takes precedence over the original sort order for the record stream. It does not change the file organization of the records. The SORTED BY clause enables you to produce reports with data records organized into groups. Certain fields will control how the organization of these reports takes place. For more information on control group reports, see the *DATATRIEVE–11 Guide to Writing Reports*.

Chapter 8

Using Compound Statements

When you want to do something in DATATRIEVE that involves definitions in a data dictionary, you usually need to use a DATATRIEVE command. When you want to manipulate data in a dictionary, you usually use DATATRIEVE statements, such as STORE, MODIFY, PRINT, and FIND.

You can enter individual statements or combine them into compound statements. You can enter statements at DATATRIEVE command level (the DTR> prompt), in procedures, or in command files.

You can also enter individual commands at DATATRIEVE command level, in procedures, or command files. However, you cannot combine DATATRIEVE commands into compound commands, mix commands and statements to form compound statements, or include commands in BEGIN-END blocks.

The *DATATRIEVE–11 Reference Manual* tells you whether a DATATRIEVE keyword is used in commands or statements.

This chapter describes the use of compound statements, REPEAT and FOR statements, and BEGIN-END blocks.

8.1 Using REPEAT to Combine Statements

Often you want to use the same DATATRIEVE statement over and over. For instance, if you are storing five new boats into the domain YACHTS, you could ready the domain for WRITE access and then repeat the instruction STORE YACHTS five times.

By using a compound statement, however, you can combine the STORE statement with a REPEAT statement and then type the STORE statement only once for the five records. The following example shows a frequent use of a compound statement—combining STORE with REPEAT:

```
DTR> READY YACHTS WRITE RET
DTR> REPEAT 3 STORE YACHTS RET
Enter MANUFACTURER: HOBIE RET
Enter MODEL: CAT RET
Enter RIG: SLOOP RET
Enter LENGTH_OVER_ALL: 22 RET
Enter DISPLACEMENT: 4000 RET
Enter BEAM: 8 RET
Enter PRICE: 6500 RET
Enter MANUFACTURER: RIDGE RET
Enter MODEL: ACT RET
Enter RIG: SLOOP RET
Enter LENGTH_OVER_ALL: 22 RET
Enter DISPLACEMENT: 3500 RET
Enter BEAM: <TAB>  RET
Enter PRICE:<TAB>  RET
Enter MANUFACTURER: ROBERTS RET
Enter MODEL: Z11 RET
Enter RIG: SLOOP RET
Enter LENGTH_OVER_ALL: 25 RET
Enter DISPLACEMENT: 4500 RET
Enter BEAM: 10 RET
Enter PRICE: 7500 RET
DTR>
```

Prompts are repeated for each field in the record until data is stored in the specified number of records or until you end the operation by entering CTRL/Z.

When you use a REPEAT statement with MODIFY, PRINT, and REPORT statements, you may also want to use a prompting value expression (*.prompt). You probably do not want to PRINT, MODIFY, or REPORT on a single record more than once. However, you can use a prompting value expression to supply new information each time a statement is repeated.

The following example uses the prompting value expression with a REPEAT loop:

```
DTR> SET NO PROMPT RET
DTR> READY YACHTS RET
DTR> REPEAT *."NUMBER OF TIMES TO REPORT" RET
CON> BEGIN RET
CON> REPORT FIRST 1 YACHTS WITH LOA = *."THE LOA" RET
RW> PRINT BOAT RET
RW> END_REPORT RET
CON> END RET
Enter NUMBER OF TIMES TO REPORT: 2 RET
Enter THE LOA: 37 RET

                                                                  23-Oct-8
                                                                  Page 1

                                          LENGTH
                                           OVER
MANUFACTURER        MODEL         RIG      ALL      WEIGHT     BEAM     PRICE

 ALBERG          37 MK II      KETCH      37        20,000      12      $36,951
```

```
Enter THE LOA: 36 RET

                                                          23-Oct-87
                                                          Page 1

                                          LENGTH
                                           OVER
MANUFACTURER        MODEL        RIG       ALL      WEIGHT     BEAM      PRICE

 CABOT              36           SLOOP     36       15,000     12

DTR>
```

A compound statement can include any DATATRIEVE statement except FIND, SELECT, DROP, RELEASE, or SORT.

Note that if you follow the REPEAT statement with a procedure, DATATRIEVE repeats only the first statement in the procedure. To repeat the complete procedure, you must use a BEGIN-END block, described in Section 8.3. Note also that a procedure in a REPEAT statement cannot include DATATRIEVE commands.

8.2 Using the FOR Statement

You can use a FOR statement when you want to access individual fields more than once. Suppose you want to supply a price for yachts with no price listed. You do not want to change all the fields in the target records. You want to change only the price field for specific yachts. First, form a collection of the boats you want to modify. Then, use a FOR statement to modify the target records:

```
DTR> SET NO PROMPT RET
DTR> READY YACHTS MODIFY RET
DTR> FIND FIRST 3 YACHTS WITH PRICE = 0 RET
[3 records found]
DTR> PRINT ALL RET

                                  LENGTH
                                   OVER
MANUFACTURER    MODEL     RIG      ALL    WEIGHT BEAM  PRICE

 BLOCK I.       40        SLOOP    39     18,500  12
 BUCCANEER      270       SLOOP    27      5,000  08
 BUCCANEER      320       SLOOP    32     12,500  10

DTR> FOR CURRENT RET
CON> MODIFY USING PRICE = DISP * 1.3 + 5000 RET
DTR> PRINT CURRENT RET
```

```
                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 BLOCK I.      40         SLOOP   39    18,500  12  $29,050
 BUCCANEER     270        SLOOP   27     5,000  08  $11,500
 BUCCANEER     320        SLOOP   32    12,500  10  $21,250

DTR>
```

8.3 Using BEGIN-END Blocks to Combine Statements

Another way to combine statements is to use a BEGIN-END block, which causes DATATRIEVE to treat several statements as one statement. BEGIN-END blocks are especially useful within FOR, STORE, and REPEAT statements.

8.3.1 BEGIN-END Blocks in FOR Statements

You can, for instance, use a BEGIN-END block in a FOR statement to modify the price field in specific records. The BEGIN-END block lets you include two PRINT statements within the MODIFY statement: the first to display the unchanged records, the second to show the records after DATATRIEVE has modified them. For example:

```
DTR> SET NO PROMPT RET
DTR> READY YACHTS WRITE RET
DTR> FOR YACHTS WITH PRICE = 0 RET
CON> MODIFY USING RET
CON> BEGIN RET
CON>     PRINT RET
CON>     PRICE = *."NEW PRICE" RET
CON>     PRINT RET
CON> END RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 BLOCK I.      40         SLOOP   39    18,500  12
Enter NEW PRICE: 29050 RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 BLOCK I.      40         SLOOP   39    18,500  12  $29,050
 BUCCANEER     270        SLOOP   27     5,000  08
Enter NEW PRICE: CTRL/Z
Execution terminated by operator
DTR>
```

8.3.2 IF-THEN-ELSE Statements in BEGIN-END Blocks

You can include an IF-THEN-ELSE statement with a prompting expression within the block to allow you to decide whether or not to modify each record stream:

```
DTR> SET NO PROMPT RET
DTR> FOR FIRST 3 YACHTS WITH PRICE = 0 RET
CON> BEGIN RET
CON>     IF *."Y TO MODIFY PRICE, N TO SKIP" CONT "Y" RET
CON>     THEN MODIFY PRICE ELSE RET
CON>     PRINT "NO CHANGE" RET
CON> END RET
Enter Y TO MODIFY PRICE, N TO SKIP: Y  RET
Enter PRICE: 23456  RET
Enter Y TO MODIFY PRICE, N TO SKIP: N  RET
NO CHANGE
Enter Y TO MODIFY PRICE, N TO SKIP: N  RET
NO CHANGE

DTR>
```

Conversely, if there are several DATATRIEVE statements required in the THEN and ELSE clauses, include them in BEGIN-END blocks.

8.3.3 BEGIN-END Blocks in STORE Statements

Often you may want to include a number of lines in the BEGIN-END block. The following example shows how to use:

- A BEGIN-END block within the STORE statement
- A prompting value expression to request user response
- A BEGIN-END block in a VERIFY clause
- A VERIFY clause with a USING clause to allow you to decide whether or not to store the record displayed in the PRINT statement
- A context variable (A) to establish DATATRIEVE context

See Appendix A for more information on DATATRIEVE context.

```
DTR> READY YACHTS WRITE RET
DTR> STORE A IN YACHTS USING RET
CON>   BEGIN RET
CON>     BUILDER = *.BUILDER RET
CON>     MODEL = *.MODEL RET
CON>     RIG = *.RIG RET
CON>     LOA = *.LENGTH RET
CON>     DISP = *.WEIGHT RET
CON>     BEAM = *.BEAM RET
CON>     PRICE = DISP * 1.3 + BEAM * 100 RET
CON>   END VERIFY USING RET
CON>   BEGIN RET
CON>     PRINT A.BOAT, SKIP RET
CON>     IF *.CONFIRMATION CONT "N" THEN RET
CON>       ABORT "BAD RECORD" RET
CON>   END RET
DTR>
```

When you press RETURN to enter the compound statement, DATATRIEVE prompts for each of the fields in the YACHTS record, then requests a confirmation:

```
Enter BUILDER: CRIS-CRAFT RET
Enter MODEL: TAB RET
Enter RIG:  KETCH RET
Enter LENGTH: 32 RET
Enter WEIGHT: 8,725 RET
Enter BEAM: 10 RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL   WEIGHT BEAM  PRICE  MANUFACTURER

 CRIS-CRAFT              KETCH   32     8,725  10  $12,343  CRIS-CRAFT

Enter CONFIRMATION: N RET
ABORT: BAD RECORD
DTR> FIND YACHTS WITH BUILDER = CRIS-CRAFT RET
[0 records found]
```

8.3.4 BEGIN-END Blocks in REPEAT Statements

If you want to repeat a sequence of statements, use a BEGIN-END block inside a REPEAT statement. Suppose you wanted to store 50 new yachts using the MANUFACTURER, LOA, DISPLACEMENT, and PRICE fields. You could use the following BEGIN-END block to repeat the sequence of prompting statements:

```
DTR> SET NO PROMPT RET
DTR> READY YACHTS WRITE RET
DTR> REPEAT 50 STORE YACHTS USING RET
CON> BEGIN RET
CON>            MANUFACTURER = *.MANUFACTURER RET
CON>            LOA          = *.LOA RET
CON>            DISPLACEMENT = *.DISPLACEMENT RET
CON>            PRICE        = *.PRICE RET
CON>  END RET
Enter MANUFACTURER: GRAMPIAN RET
Enter LOA: 40 RET
Enter DISPLACEMENT: 1400 RET
Enter PRICE: 23456 RET
Enter MANUFACTURER: HIGGINS RET
Enter LOA: 37 RET
Enter DISPLACEMENT: 1375 RET
Enter PRICE: 14765 RET
       .
       .
       .
DTR>
```

For information on invoking a procedure in a REPEAT statement, see Chapter 9.

Because statements that use BEGIN-END blocks can be quite long, it is often useful to put them into DATATRIEVE procedures so that you can edit and reuse them without having to retype them.

Chapter 9

Using DATATRIEVE Procedures

Often you want to execute the same series of commands and statements over and over again, and you may want to have other users execute those same commands and statements. Unless you use procedures, you have to retype the input each time. By using procedures, however, you can develop the series of steps once and then simply invoke the procedure each time you want to do the same steps over again.

A procedure is a fixed sequence of DATATRIEVE commands and statements you create, name, and store in your data dictionary. A procedure can also contain portions of a command or statement, such as a complex value expression.

9.1 Defining a Procedure

For almost any series of statements you use over and over again, you can save yourself time by defining a single procedure. For example, you repeatedly perform a simple query to display information about the manufacturers of large yachts:

```
DTR> READY YACHTS RET
DTR> FIND BIGGIES IN YACHTS WITH LOA GT 40 SORTED BY BUILDER RET
[8 records found]
DTR> PRINT ALL RET
```

```
                                LENGTH
                                 OVER
MANUFACTURER    MODEL      RIG    ALL    WEIGHT BEAM  PRICE

 CHALLENGER  41          KETCH   41     26,700  13  $51,228
 COLUMBIA    41          SLOOP   41     20,700  11  $48,490
 GULFSTAR    41          KETCH   41     22,000  12  $41,350
 ISLANDER    FREEPORT    KETCH   41     22,000  13  $54,970
 NAUTOR      SWAN 41     SLOOP   41     17,750  12
 NEWPORT     41 S        SLOOP   41     18,000  11
 OLYMPIC     ADVENTURE   KETCH   42     24,250  13  $80,500
 PEARSON     419         KETCH   42     21,000  13

DTR>
```

Rather than type this query repeatedly, you can put it into a procedure. To define a procedure, enter the DEFINE PROCEDURE command at DATATRIEVE command level. After typing the keywords DEFINE PROCEDURE, enter a name for the procedure and press RETURN:

```
DTR> DEFINE PROCEDURE BUILDERS RET
```

DATATRIEVE then prompts with DFN> to indicate that it expects a procedure definition. Enter the commands or statements that form the procedure definition. DATATRIEVE continues to prompt with DFN> until you enter the keyword END_PROCEDURE on a line by itself.

```
DTR> DEFINE PROCEDURE BUILDERS RET
DFN>          .
DFN>          .
DFN>          .
DFN>          END_PROCEDURE RET
DTR>
```

As soon as you enter END_PROCEDURE, DATATRIEVE stores the procedure definition in your current dictionary. It checks the syntax of the DEFINE PROCEDURE statement, but not the syntax of the statements the procedure contains. DATATRIEVE checks for syntax errors in those statements only when you invoke the procedure.

9.2 Invoking a Procedure

You invoke a procedure by preceding its name with a colon:

```
:procedure-name
```

The content of a procedure determines where you can invoke it. In general, you can invoke a procedure anywhere you can use the commands or statements contained in the procedure. For example, if the procedure contains only complete DATATRIEVE commands and statements, you can invoke it at the DATATRIEVE command level:

```
DTR> :BUILDERS RET
```

Note that you cannot invoke a procedure during an ADT, EDIT, or GUIDE Mode session or in a domain, record, or table definition.

In addition, when DATATRIEVE executes a procedure, you do not see the commands and statements in the procedure or the system messages that are normally displayed. You see only the output that follows the last statement or command in the procedure. In the following example, the only output is in response to FIND YACHTS SORTED BY DESC PRICE, the last statement in the procedure EXPENSIVE:

```
DTR> SHOW EXPENSIVE RET
PROCEDURE EXPENSIVE
READY YACHTS
FIND YACHTS SORTED BY DESC PRICE
END_PROCEDURE
DTR> :EXPENSIVE RET
[113 records found]
DTR>
```

If you follow the FIND statement in the command procedure with another statement, you no longer receive the message about the number of records found when you invoke the procedure:

```
DTR> SHOW EXPENSIVE RET
PROCEDURE EXPENSIVE
READY YACHTS
FIND YACHTS SORTED BY DESC PRICE
PRINT FIRST 5 CURRENT
END_PROCEDURE
DTR> :EXPENSIVE RET

                                LENGTH
                                 OVER
MANUFACTURER'   MODEL      RIG    ALL    WEIGHT BEAM  PRICE

 OLYMPIC      ADVENTURE   KETCH   42     24,250  13   $80,500
 ISLANDER     FREEPORT    KETCH   41     22,000  13   $54,970
 CHALLENGER   41          KETCH   41     26,700  13   $51,228
 NORTHERN     37          KETCH   37     14,000  11   $50,000
 COLUMBIA     41          SLOOP   41     20,700  11   $48,490

DTR>
```

9.3 Contents of a Procedure

A procedure can contain any number of the following DATATRIEVE elements:

- Full DATATRIEVE commands and statements
- Command and statement clauses and arguments
- Comments

9.3.1 Commands and Statements in Procedures

You might define a procedure containing complete DATATRIEVE commands and statements. The following procedure definition, for instance, finds and displays the biggest yachts in the domain:

```
DTR> DEFINE PROCEDURE BIG_YACHTS RET
DFN> FIND BIGGIES IN YACHTS WITH LOA GT 40 SORTED BY BUILDER RET
DFN> PRINT ALL RET
DFN> END_PROCEDURE RET
DTR>
```

When you execute the procedure BIG_YACHTS, the results are the same as entering the FIND and PRINT statements at the DATATRIEVE command level, indicated by the DTR> prompt:

```
DTR> READY YACHTS RET
DTR> :BIG_YACHTS RET

                                LENGTH
                                OVER
MANUFACTURER    MODEL     RIG   ALL   WEIGHT BEAM  PRICE

 CHALLENGER  41          KETCH  41    26,700  13  $51,228
 COLUMBIA    41          SLOOP  41    20,700  11  $48,490
 GULFSTAR    41          KETCH  41    22,000  12  $41,350
 ISLANDER    FREEPORT    KETCH  41    22,000  13  $54,970
 NAUTOR      SWAN 41     SLOOP  41    17,750  12
 NEWPORT     41 S        SLOOP  41    18,000  11
 OLYMPIC     ADVENTURE   KETCH  42    24,250  13  $80,500
 PEARSON     419         KETCH  42    21,000  13

DTR>
```

9.3.2 Arguments and Clauses

Besides full commands and statements, a procedure can contain fragments of statements or commands. It can contain an argument or clause from a command or statement. For example, a procedure can contain a record selection expression:

```
DTR> DEFINE PROCEDURE BIG_YACHTS_RSE RET
DFN> BIGGIES IN YACHTS WITH LOA GT 40 SORTED BY BUILDER RET
DFN> END_PROCEDURE RET
DTR>
```

Having separated the record selection expression from the FIND statement, you can then use the procedure name as the argument of a FIND statement:

```
DTR> FIND :BIG_YACHTS_RSE RET
[8 records found]
DTR>
```

In fact, you can use this procedure in any command or statement containing an RSE argument, such as the PRINT statement:

```
DTR> PRINT ALL :BIG_YACHTS_RSE RET
```

MANUFACTURER	MODEL	RIG	LENGTH OVER ALL	WEIGHT	BEAM	PRICE
CHALLENGER	41	KETCH	41	26,700	13	$51,228
COLUMBIA	41	SLOOP	41	20,700	11	$48,490
GULFSTAR	41	KETCH	41	22,000	12	$41,350
ISLANDER	FREEPORT	KETCH	41	22,000	13	$54,970
NAUTOR	SWAN 41	SLOOP	41	17,750	12	
NEWPORT	41 S	SLOOP	41	18,000	11	
OLYMPIC	ADVENTURE	KETCH	42	24,250	13	$80,500
PEARSON	419	KETCH	42	21,000	13	

You can begin a procedure with the end fragment of a command or statement and include other whole commands or statements. You can also end a procedure with the beginning fragment of a command or statement after a series of complete commands and statements.

9.3.3 Comments in Procedures

A comment contains explanatory information for you or other users that DATATRIEVE does not interpret as input. To put a comment in a procedure, put an exclamation point (!) before the information that you want to include.

When you invoke a procedure, DATATRIEVE processes it without displaying the contents of the procedure. To display the comments, use the SHOW command with the procedure name. You could, for instance, put a comment into the procedure BIG_YACHTS_QUERY. The results of the procedure are the same as without the comment, but when you use a SHOW command, you can see the explanatory comment:

```
DTR> SHOW BIG_YACHTS_QUERY RET
SET ABORT
DECLARE LENGTH PIC 99
VALID IF LENGTH GT 35.
!
!THIS PROCEDURE SHOWS YACHTS GE SPECIFIED LENGTH
!
LENGTH = *."MIN LOA"
IF LENGTH GT 42
THEN ABORT "NO BOATS THAT BIG"
FIND BIGGIES IN YACHTS WITH LOA GE LENGTH SORTED BY BUILDER
PRINT BUILDER, RIG, LOA, PRICE OF BIGGIES
END_PROCEDURE
DTR>
```

9.4 Using Procedures to Locate Errors

When you invoke a procedure, DATATRIEVE processes the contents of the procedure. If it finds an error, it issues a message. Suppose, for instance, you created the following long procedure:

```
DTR> DEFINE PROCEDURE WAGE_REPORT RET
DFN> REPORT WAGES RET
DFN> SET REPORT_NAME = WEEKLY WAGE REPORT RET
DFN> SET COLUMNS_PAGE = 70 RET
DFN> PRINT LAST_NAME, GROSS_PAY, FICA, RET
DFN> FEDERAL_TAX, STATE_TAX, RET
DFN> GROSS_PAY - (FICA + FEDERAL_TAX + STATE_TAX)*("NET PAY") USING RET
DFN> $$,$$$.99 RET
DFN> AT BOTTOM OF REPORT PRINT SKIP 2, COL 1, "TOTAL:", RET
DFN> TOTAL GROSS_PAY USING $$$,$$$.99, RET
DFN> TOTAL FICA USING $$$,$$$.99, RET
DFN> TOTAL FEDERAL_TAX USING $$$,$$$.99, RET
DFN> TOTAL STATE_TAX USING $$$,$$$.99, RET
DFN> TOTAL (GROSS_PAY - (FICA + FEDERAL_TAX + STATE_TAX)) USING RET
DFN> $$$,$$$.99 RET
DFN> END_REPORT RET
DFN> END_PROCEDURE RET
DTR>
```

If you have made any errors, DATATRIEVE stops executing when it finds the first error and sends you an error message, as in this example:

```
DTR> :WAGE_REPORT RET
Invalid column header or report name (WEEKLY)
DTR> EDIT WAGE_REPORT RET
```

Edit the procedure to place quotation marks around "WEEKLY WAGE REPORT". Then try the procedure again:

```
DTR> :WAGE_REPORT RET
Field "WAGES" is undefined or used out of context
DTR> EDIT WAGE_REPORT RET
```

To correct this second error, edit the procedure to place the READY WAGES command before the REPORT statement. Then invoke the procedure again:

```
DTR> :WAGE_REPORT RET
Enter COLUMNS PER PAGE: 80 RET

                              WEEKLY WAGE REPORT                    23-Oct-87
                                                                    Page 1

      LAST          GROSS                        FEDERAL      STATE
      NAME           PAY          FICA             TAX         TAX          NET PAY

   BLAKE          $1,000.00      $103.86         $204.77        $.01        $691.36
   DUNN           $1,500.00      $145.87         $297.98      $54.32      $1,001.83
   HILL             $500.00       $52.93          $79.75      $32.98        $334.34
   CHONTZ           $999.99      $103.85         $204.76      $57.90        $633.48
   MOONY          $1,900.98      $145.87         $375.98      $75.90      $1,303.23
   STARK          $9,500.00      $145.87         $999.84     $106.90      $8,247.39

TOTAL:           $15,400.97      $698.25       $2,163.08     $328.01     $12,211.63

DTR>
```

9.5 A Sample Procedure

You can create your own procedures now using the following example as a model. The following is a procedure that uses the Report Writer to write a summary report of yacht data:

```
DTR> DEFINE PROCEDURE YACHT_SUMMARY RET
DFN> SET ABORT RET
DFN> PRINT "THIS REPORT REQUIRES AN ESTABLISHED COLLECTION," RET
DFN> PRINT "SORTED BY LOA AND BEAM." RET
DFN> PRINT "HAVE YOU ESTABLISHED A COLLECTION?" RET
DFN> IF *."YES OR NO" CONTAINING "N" THEN ABORT "COLLECTION NEEDED." RET
DFN> REPORT ON *."OUTPUT DEVICE OR FILE" RET
DFN> SET REPORT_NAME="EXAMPLE: REPORT FROM A PROCEDURE" RET
DFN> SET LINES_PAGE=55, COLUMNS_PAGE=60 RET
DFN> PRINT BUILDER, MODEL, LOA, BEAM, PRICE RET
DFN> AT BOTTOM OF LOA PRINT SKIP, COL 30 "AVERAGE PRICE =", RET
DFN>   AVERAGE (PRICE), SKIP RET
DFN> AT BOTTOM OF REPORT PRINT COL 17,"NUMBER OF BOATS = ", RET
DFN>   COL 35, COUNT, SKIP, "AVERAGE PRICE OF ALL BOATS =", RET
DFN> AVERAGE (PRICE) RET
DFN> END_REPORT RET
DFN> END_PROCEDURE RET
DTR>
```

This example illustrates some statements that are particularly useful in procedures:

- Use the PRINT statement to display a message when the procedure is invoked.
- The prompting value expression *."YES OR NO" requires a response to the question: HAVE YOU ESTABLISHED A COLLECTION? The Boolean expression CONTAINING checks the user's response to the question. If the response is N or NO, the procedure aborts.
- If you invoke YACHT_SUMMARY and answer NO to the first prompt, the procedure aborts:

```
DTR> :YACHT_SUMMARY RET
THIS REPORT REQUIRES AN ESTABLISHED COLLECTION,
SORTED BY LOA AND BEAM.
HAVE YOU ESTABLISHED A COLLECTION?
Enter YES OR NO: NO RET
ABORT: COLLECTION NEEDED
DTR>
```

- If you answer YES to the first prompt, but you do not actually have a current collection, the Report Writer aborts the procedure and prints an error message:

```
DTR> :YACHT_SUMMARY RET
THIS REPORT REQUIRES AN ESTABLISHED COLLECTION,
SORTED BY LOA AND BEAM.
HAVE YOU ESTABLISHED A COLLECTION?
Enter YES OR NO: YES RET
A current collection has not been established.
DTR>
```

- The prompting value expression *."OUTPUT DEVICE OR FILE" allows you to select the device or file to contain the report when DATATRIEVE executes the procedure.

- If you make a collection of YACHTS with LOA between (and including) 36 and 37 and price not equal to 0, DATATRIEVE displays the following report on your terminal:

```
DTR> READY YACHTS RET
DTR> FIND YACHTS WITH LOA BETWEEN 36 37 AND PRICE NE 0 RET
[5 records found]
DTR> SORT CURRENT BY LOA, BEAM RET
DTR> :YACHT_SUMMARY RET
THIS REPORT REQUIRES AN ESTABLISHED COLLECTION,
SORTED BY LOA AND BEAM.
HAVE YOU ESTABLISHED A COLLECTION?
Enter YES OR NO: YES RET
Enter OUTPUT DEVICE OR FILE: TI: RET

          EXAMPLE: REPORT FROM A PROCEDURE    01-Apr-1987
                                              Page 1

                                  LENGTH
                                   OVER
 MANUFACTURER          MODEL       ALL       BEAM      PRICE

   ISLANDER          36             36         11      $31,730
   I. TRADER         37             36         12      $39,500

 AVERAGE PRICE =                                       $35,615

   IRWIN             37 MARK II     37         11      $36,950
   NORTHERN          37             37         11      $50,000
   ALBERG            37 MK II       37         12      $36,951

 AVERAGE PRICE =                                       $41,300

                NUMBER OF BOATS =    5
                AVERAGE PRICE OF ALL BOATS =          $39,026
```

9.6 Nesting Procedures

A nested procedure is a procedure within another procedure.

The following procedure calculates the price per pound of a boat and assigns a column header and edit string for that value expression:

```
DTR> DEFINE PROCEDURE PRICE_PER_POUND RET
DFN> PRICE/DISPLACEMENT ("PRICE"/"PER"/"POUND") USING $$9.99 RET
DFN> END_PROCEDURE RET
DTR>
```

You cannot invoke this procedure by itself, but you can invoke the PRICE_PER_POUND procedure in another procedure that prints the builder, model, and price per pound of all boats in the CURRENT collection, as follows:

```
DTR> DEFINE PROCEDURE PRICE_REPORT RET
DFN> PRINT ALL BUILDER, MODEL, :PRICE_PER_POUND RET
DFN> END_PROCEDURE RET
DTR>
```

When you invoke the procedure PRICE_REPORT, DATATRIEVE displays three fields for each YACHTS record. First the builder and model are displayed as a result of the procedure's PRINT statement. Then the PRICE_PER_POUND procedure is called to compute and format the price/displacement before it is displayed.

The following example uses the BIG_YACHTS procedure to establish the CURRENT collection and PRICE_REPORT to print a short report:

```
DTR> :BIG_YACHTS; :PRICE_REPORT RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 CHALLENGER  41          KETCH   41    26,700  13  $51,228
 COLUMBIA    41          SLOOP   41    20,700  11  $48,490
 GULFSTAR    41          KETCH   41    22,000  12  $41,350
 ISLANDER    FREEPORT    KETCH   41    22,000  13  $54,970
 NAUTOR      SWAN 41     SLOOP   41    17,750  12
 NEWPORT     41 S        SLOOP   41    18,000  11
 OLYMPIC     ADVENTURE   KETCH   42    24,250  13  $80,500
 PEARSON     419         KETCH   42    21,000  13

                         PRICE
                          PER
MANUFACTURER    MODEL    POUND

 CHALLENGER  41           $1.92
 COLUMBIA    41           $2.34
 GULFSTAR    41           $1.88
 ISLANDER    FREEPORT     $2.50
 NAUTOR      SWAN 41      $0.00
 NEWPORT     41 S         $0.00
 OLYMPIC     ADVENTURE    $3.32
 PEARSON     419          $0.00

DTR> EXIT RET
```

When nesting procedures, do not let a procedure invoke itself. You can create an infinite loop. (Should you create such a loop, press CTRL/C two times to stop your process.)

9.7 Using a Procedure in a Compound Statement

To execute a procedure a number of times, you can invoke it within a REPEAT or FOR statement. You should be careful when invoking a procedure in these statements, however. For example, the following procedure appears to be correct but produces unexpected results:

```
DTR> SHOW EX1 RET
PROCEDURE EX1
FOR
   YACHTS WITH PRICE = 0 AND LOA BT 16 AND 23
     PRINT BUILDER, MODEL, LOA
PRINT "Printing Test Record"
END_PROCEDURE
DTR> REPEAT 3 :EX1
                          LENGTH
                           OVER
MANUFACTURER    MODEL      ALL
 ENCHILADA    20           20
 ERICSON      23/ SPECIA   23
 SAN JUAN     21           21
 ENCHILADA    20           20
 ERICSON      23/ SPECIA   23
 SAN JUAN     21           21
 ENCHILADA    20           20
 ERICSON      23/ SPECIA   23
 SAN JUAN     21           21
Printing Test Record
```

Only the FOR statement in the EX1 procedure is repeated. The PRINT statement is executed only once at the very end. When DATATRIEVE encounters the first complete statement in a procedure, it assumes that the REPEAT statement is also complete. Therefore, it repeats only the first statement in the procedure and executes each of the remaining statements once.

To repeat the entire procedure, enclose the procedure call or the procedure definition in a BEGIN-END block. For example, the following sequence of statements puts a procedure in a BEGIN-END block and repeats the procedure three times:

```
DTR> REPEAT 3 BEGIN RET
[Looking for statement]
CON> :EX1 RET
CON> END RET
```

```
                          LENGTH
                           OVER
MANUFACTURER   MODEL      ALL
 ENCHILADA   20           20
 ERICSON     23/ SPECIA   23
 SAN JUAN    21           21
Printing Test Record
 ENCHILADA   20           20
 ERICSON     23/ SPECIA   23
 SAN JUAN    21           21
Printing Test Record
 ENCHILADA   20           20
 ERICSON     23/ SPECIA   23
 SAN JUAN    21           21
Printing Test Record
```

The following example uses a FOR statement, includes a BEGIN-END block in the procedure, and invokes the procedure in a REPEAT statement:

```
DTR> SHOW EX3 RET
PROCEDURE EX3
FOR
   YACHTS WITH PRICE = 0 AND LOA BT 16 AND 23
   BEGIN
      PRINT BUILDER, MODEL, LOA
      PRINT "Print Test Record"
   END
END_PROCEDURE

DTR> REPEAT 3 :EX3 RET
                          LENGTH
                           OVER
MANUFACTURER   MODEL      ALL
 ENCHILADA   20           20
Print Test Record
 ERICSON     23/ SPECIA   23
Print Test Record
 SAN JUAN    21           21
Print Test Record
 ENCHILADA   20           20
Print Test Record
 ERICSON     23/ SPECIA   23
Print Test Record
 SAN JUAN    21           21
Print Test Record
 ENCHILADA   20           20
Print Test Record
 ERICSON     23/ SPECIA   23
Print Test Record
 SAN JUAN    21           21
Print Test Record

DTR>
```

If you invoke a procedure in a FOR statement, you must use the same technique. Enclose the call or the procedure definition in a BEGIN-END block as in the following example:

```
DTR> SHOW PRICE_REPORT2 RET
PROCEDURE PRICE_REPORT2
PRINT BUILDER, MODEL, :PRICE_PER_POUND
END_PROCEDURE

DTR>

DTR> FOR YACHTS WITH PRICE GT 20000 AND LOA LT 29 RET
[Looking for statement]
CON> BEGIN RET
[Looking for statement]
CON> :PRICE_REPORT2 RET
[Looking for statement or "END"]
CON> END RET

                          PRICE
                          PER
MANUFACTURER     MODEL    POUND

 CAPE DORY       28       $2.44
 SABRE           28       $2.97

DTR>
```

Remember that if you use a procedure in a loop, do not include a FIND, SELECT, DROP, or RELEASE statement. These statements cannot appear in BEGIN-END blocks.

9.8 Aborting Procedures

You can abort a procedure by including a SET ABORT statement in the procedure definition. If the abort conditions arise and SET ABORT is in effect, DATATRIEVE aborts the procedure and prints a message on your terminal. If SET NO ABORT is in effect, DATATRIEVE aborts the command or statement that contains the ABORT but continues to execute the other commands and statements in the procedure.

The default setting in DATATRIEVE is SET ABORT. You can ensure that SET ABORT is in effect by including that statement in the procedure definition:

```
DTR> DEFINE PROCEDURE BIG_YACHTS_QUERY RET
DFN> SET ABORT RET
DFN> DECLARE LENGTH PIC 99 RET
DFN>   VALID IF LENGTH GT 35. RET
DFN> LENGTH = *."MIN LOA" RET
DFN> IF LENGTH GT 42 RET
DFN>   THEN ABORT "NO BOATS THAT BIG" RET
DFN> FIND BIGGIES IN YACHTS WITH LOA GE LENGTH RET
DFN>   SORTED BY BUILDER RET
DFN> PRINT BUILDER, RIG, LOA, PRICE OF BIGGIES RET
DFN> END_PROCEDURE RET
DTR>
```

If you invoke BIG_YACHTS_QUERY and supply a length of 35 or smaller, DATATRIEVE reprompts you for a valid length. If you supply a length greater than 42, the procedure aborts, prints the specified abort message, and returns you to DATATRIEVE command level:

```
DTR> :BIG_YACHTS_QUERY RET
Enter MIN LOA: 35 RET
Validation error for LENGTH
Re-enter MIN LOA: 43 RET
ABORT: NO BOATS THAT BIG
Execution terminated by "ABORT" statement
DTR>
```

If you assign a value between 36 and 42 to length, DATATRIEVE prints the appropriate collection:

```
DTR> :BIG_YACHTS_QUERY RET
Enter MIN LOA: 39 RET

                    LENGTH
                     OVER
MANUFACTURER  RIG    ALL    PRICE

 BLOCK I.     SLOOP   39
 CHALLENGER   KETCH   41    $51,228
 COLUMBIA     SLOOP   41    $48,490
 GULFSTAR     KETCH   41    $41,350
 ISLANDER     KETCH   41    $54,970
 LINDSEY      MS      39    $35,900
 NAUTOR       SLOOP   41
 NEWPORT      SLOOP   41
 OLYMPIC      KETCH   42    $80,500
 PEARSON      SLOOP   39
 PEARSON      KETCH   42
```

9.9 Displaying Procedure Information

You can list the names of all procedures in your default directory with the SHOW command:

```
DTR> SHOW PROCEDURES RET
Procedures:
        BIG             BIG_YACHTS_QUERY                CHEAP
        MS_SEARCH       PHONE_REP           TEST        YACHT_SUMMARY

DTR>
```

If you want to display a procedure on your terminal, you can use the SHOW command and specify the name of the procedure to be displayed:

```
DTR> SHOW MS_SEARCH RET
PROCEDURE MS_SEARCH
READY YACHTS
FIND YACHTS WITH RIG = "MS"
FOR CURRENT PRINT BUILDER,
(BUILDER VIA COMPANY_TABLE) ("ADDRESS")
END_PROCEDURE

DTR>
```

9.10 Editing Procedures

You can correct an error with the EDIT command. Invoke EDT using the following format at the DTR> prompt:

```
EDIT procedure-name
```

When you find the error, use EDT to correct it or follow this sequence:

1. EXTRACT the procedure from DATATRIEVE to a command file on your system using the following statement:

   ```
   EXTRACT ON file-spec procedure-name
   ```

   Use the file extension .CMD in the file specification to distinguish the extracted file as a command file.
2. Exit from DATATRIEVE.
3. Use the text editor you normally use on your system to revise the command file containing the procedure.
4. Return to DATATRIEVE.
5. Invoke the command file by typing the at sign (@) and the name of the file to bring the corrected procedure into DATATRIEVE.

See Chapter 10 for information on DATATRIEVE command files and Chapter 16 in this manual for information on the EDIT command that invokes EDT. See EDT documentation for full information on the EDT editing language.

9.11 Deleting Procedures

You can delete a procedure from your dictionary with the DELETE command:

```
DTR> SHOW PROCEDURES RET
Procedures:
        BIG                 BIG_YACHTS_QUERY        CHEAP
        MS_SEARCH           PHONE_REP               YACHT_SUMMARY

DTR> DELETE BIG; RET
DTR> SHOW PROCEDURES RET
Procedures:
        BIG_YACHTS_QUERY            CHEAP               MS_SEARCH
        PHONE_REP                   YACHT_SUMMARY

DTR>
```

Note that the DELETE command must end with a semicolon (;).

You should maintain a backup copy of your procedure, especially if it is a long one. Use the DATATRIEVE EXTRACT command to copy your procedure to a command file for backup. Once you have a backup copy, you can always recover the procedure if you happen to delete it accidentally.

The following example illustrates how you can create a backup file, delete a procedure, and replace it without ever leaving DATATRIEVE:

```
DTR> SET COLUMNS_PAGE = 80 RET
DTR> SHOW PROCEDURES RET
Procedures:
        BIG_YACHTS          BIG_YACHTS_QUERY                    BP
        BREAK_REP           CTRL                EXPENSIVE       F
        FAM_REC             INFLATION_REPORT                    JOB_HISTORY
        MULTIPLE_PRINT      MULTIPLE_STORE      NEW_YACHTS      NME
        P                   PAGE_HEADER         PICKBOATS       PRICE_INCREASE
        R                   SALARY_REPORT       SALARY_REPORT1  SALARY_REPORT2
        SALARY_TOTALS       SUM                 TAB_TEST        TEST_WAGE
        TITLE_PAGE          V                   WAGE_REPORT     WP
        XP                  YACHTS_REPORT       YACHT_PER_LB    YACHT_PRICE
DTR> EXTRACT ON SAVBIG BIG_YACHTS_QUERY RET
DTR> DELETE BIG_YACHTS_QUERY; RET
DTR> SHOW PROCEDURES RET
```

```
Procedures:
        BIG_YACHTS      BP              BREAK_REP       CTRL
        EXPENSIVE       F               FAM_REC         INFLATION_REPORT
        JOB_HISTORY     MULTIPLE_PRINT  MULTIPLE_STORE  NEW_YACHTS
        NME             P               PAGE_HEADER     PICKBOATS
        PRICE_INCREASE  R               SALARY_REPORT   SALARY_REPORT1
        SALARY_REPORT2  SALARY_TOTALS   SUM             TAB_TEST
        TEST_WAGE       TITLE_PAGE      V               WAGE_REPORT
        WP              XP              YACHTS_REPORT   YACHT_PER_LB
        YACHT_PRICE
DTR> @SAVBIG
DELETE BIG_YACHTS_QUERY;
"BIG_YACHTS_QUERY" has not been defined in the dictionary
DEFINE PROCEDURE BIG_YACHTS_QUERY
SET ABORT
DECLARE LENGTH PIC 99
VALID IF LENGTH GT 35.
LENGTH = *."MIN LOA"
IF LENGTH GT 42
THEN ABORT "NO BOATS THAT BIG"
FIND BIGGIES IN YACHTS WITH LOA GE LENGTH SORTED BY BUILDER
PRINT BUILDER, RIG, LOA, PRICE OF BIGGIES
END_PROCEDURE
DTR> SHOW PROCEDURES RET
Procedures:
        BIG_YACHTS      BIG_YACHTS_QUERY                BP
        BREAK_REP       CTRL            EXPENSIVE       F
        FAM_REC         INFLATION_REPORT                JOB_HISTORY
        MULTIPLE_PRINT  MULTIPLE_STORE  NEW_YACHTS      NME
        P               PAGE_HEADER     PICKBOATS       PRICE_INCREASE
        R               SALARY_REPORT   SALARY_REPORT1  SALARY_REPORT2
        SALARY_TOTALS   SUM             TAB_TEST        TEST_WAGE
        TITLE_PAGE      V               WAGE_REPORT     WP
        XP              YACHTS_REPORT   YACHT_PER_LB    YACHT_PRICE
DTR> :BIG_YACHTS_QUERY RET
Enter MIN LOA: CTRL/Z
Execution terminated by operator
DTR>
```

Chapter 10

Using DATATRIEVE Command Files

Many people use DATATRIEVE by typing in single commands in the form @ENTERC or :MONTHLY_REPORT and then watching the results scroll on their screens. Someone else has prepared a command file or procedure for them. Within the command file or procedure are the DATATRIEVE commands and statements to carry out a given task.

Command files are much like procedures. Both contain fixed sequences of DATATRIEVE commands and statements and both allow you to execute frequently used operations. They have the following differences:

- You invoke a command file by typing the at sign (@) before the file specification, a procedure by typing a colon (:) before the procedure name.
- You store the procedures in your dictionary. You can see your procedures with a SHOW PROCEDURES command from within DATATRIEVE. Your command files, on the other hand, reside outside DATATRIEVE in your operating system directory. You must exit from DATATRIEVE and use operating system commands to display them.
- You edit procedures with the EDIT command (which invokes EDT). You edit command files with your preferred text editor.
- When you invoke command files, you see the command statements and comments echo on your terminal.

You can use command files for the following purposes:

- To create a startup command file that will automatically execute certain commands and statements each time you invoke DATATRIEVE. Default name for the file is QUERY.INI. See Chapter 2.
- To create and then invoke command files to add definitions of dictionary objects to your dictionary.

- To use as backup files of your dictionary. If something happens to corrupt the dictionary and you need to restore the definitions it previously contained, you can use your backup files of domain, record, table, and procedure definitions.

  DATATRIEVE executes the command file and returns to your operating system's command level.
- To process files in batch mode. You can include invocation command lines to execute DATATRIEVE commands and statements.
- To develop and test a procedure you want to store in your dictionary:
  1. Put the steps of the procedure into a command file. Do not yet include the DEFINE PROCEDURE command.
  2. Execute the command file from your operating system's command level.
  3. The steps in the command file will appear on the screen and stop at the point where an error occurs. Use the message to help you decide what is wrong with the series of statements you have entered.
  4. When you have eliminated all errors from the command file, insert a DEFINE PROCEDURE statement at the beginning of the file and an END_PROCEDURE statement at the end of the file. If you have an earlier version of the procedure or another element with the same name that you do not need any more, insert a DELETE command and the file name before the DEFINE PROCEDURE statement. Be careful not to delete anything important, however.
  5. Execute the command file by typing an at sign (@) and the file name to load the procedure into your dictionary.

10.1 Creating a Command File

You create a command file at the operating system level with a text editor. Invoke an editor and enter the sequence of DATATRIEVE commands and statements just as you would in DATATRIEVE. Do not include DATATRIEVE prompts such as DTR>, CON>, DFN>, or RW>, only the commands and statements you enter following a prompt. When you complete the sequence of commands and statements and exit from the editor, your operating system stores the command file in your system directory.

It is usually a good idea to specify .CMD as the file extension of a DATATRIEVE command file. Because it is the default file extension, you do not have to type the .CMD extension when you invoke the command

file in DATATRIEVE. For example, to invoke the command file HELLO.CMD in DATATRIEVE, you can type the following:

```
DTR> @HELLO
```

10.2 Contents of a Command File

A command file can contain any DATATRIEVE command or statement, including commands that place you at other command levels. In addition, comments are recommended to document the actions of the command file.

10.2.1 ADT, EDIT, SET GUIDE

You can include ADT, EDIT, or SET GUIDE commands in a command file. DATATRIEVE places you in ADT, EDT, or Guide Mode and displays an ADT, EDT, or Guide prompt. You cannot include responses to prompts in your command file, however. The file can contain only commands or statements.

DATATRIEVE executes the next line in the command file only after you exit from EDT, Guide Mode, or ADT. Even if that line is a valid response to an ADT, EDT, or Guide Mode prompt, DATATRIEVE displays an error message and returns you to DATATRIEVE command level unless it is a valid DATATRIEVE command or statement.

10.2.2 Comments in a Command File

You can also include comments in a command file by placing an exclamation point (!) before each comment line. Comments echo on your terminal when you invoke the file.

If your command file defines a procedure and you put comments into the procedure definition, DATATRIEVE stores the comments in the dictionary along with the rest of the definition. DATATRIEVE displays those comments when you invoke the command file. When you invoke the procedure, however, DATATRIEVE does not display the comments on your terminal.

10.3 Invoking a Command File

To invoke a command file that you have catalogued in your directory, precede the file specification with an at sign (@). To invoke a command file, enter the invocation on a line by itself. For RSTS systems, use the following format:

```
@device:[PPN]filename.cmd
```

For RSX systems, use the following format:

```
@device:[UIC]filename.cmd;version
```

If the file extension is .CMD (the default) and the file is in your default directory, only the file name is needed:

```
@filename
```

10.3.1 Invoking Command Files from the System Prompt

You need not enter DATATRIEVE to invoke a DATATRIEVE command file; you can invoke it from the system level by preceding the command file invocation with DTR. For example, to invoke PRT.CMD in your operating system directory, type the following in response to the system level prompt (>):

```
Ready
> DTR @PRT RET
```

Invoking a command file in this way differs from invoking one in response to the DTR> prompt. DATATRIEVE executes all the commands and statements in the command file as though you had entered them interactively. However, it does not show the DTR> prompt on your terminal, and after executing the last command or statement in the file, it automatically exits from DATATRIEVE. If the command file is in another user's directory, you invoke it by specifying all the necessary information in the following format:

```
@device:[UIC]filename.extension
```

For RSX systems, you can also specify a version number after the extension.

10.3.2 Invoking a Command File from a Procedure

You can invoke a command file from a procedure that you define with the DEFINE PROCEDURE command. For example, suppose you create a procedure PICKBOATS to form a collection of boats that cost more than $10,000. Invoke a command file SAMPLE.CMD containing report-generating statements within PICKBOATS to produce a report:

```
DTR> DEFINE PROCEDURE PICKBOATS RET
DFN> READY YACHTS RET
DFN> FIND YACHTS WITH PRICE GT 10000 SORTED BY LOA, BEAM RET
DFN> @SAMPLE RET
SET ABORT
!THIS REPORT REQUIRES AN ESTABLISHED COLLECTION,
!SORTED BY LOA AND BEAM.
!
!HAVE YOU ESTABLISHED A COLLECTION?
IF *."YES OR NO" CONTAINING "N" THEN ABORT "SORRY, NO COLLECTION."
REPORT ON *."OUTPUT DEVICE OR FILE"
SET REPORT_NAME="SAMPLE REPORT"/"FROM A PROCEDURE"
SET LINES_PAGE=55, COLUMNS_PAGE=60
PRINT BUILDER, MODEL, LOA, BEAM, PRICE
AT BOTTOM OF LOA PRINT SKIP, "AVERAGE PRICE =",
      AVERAGE (PRICE), SKIP
AT BOTTOM OF REPORT PRINT COL 17, "NUMBER OF BOATS = ",
  COL 40, COUNT, SKIP, "AVERAGE PRICE OF ALL BOATS =", AVERAGE (PRICE)
END_REPORT
DFN> END_PROCEDURE RET
DTR> :PICKBOATS RET
Enter YES OR NO: Y RET
Enter OUTPUT DEVICE OR FILE: TI: RET
DTR>

                    SAMPLE REPORT                    25-Aug-82
                    FROM A PROCEDURE                   Page 1

                                        LENGTH
                                         OVER
  MANUFACTURER          MODEL           ALL        BEAM      PRICE

    EASTWARD            HO              24         09       $15,900

AVERAGE PRICE =                                              $15,900

    IRWIN               25              25         12       $10,950

AVERAGE PRICE =                                              $10,950

    AMERICAN            26-MS           26         08       $18,895
    GRAMPIAN            26              26         08       $11,495
    WESTERLY            CENTAUR         26         08       $15,245
    TANZER              26              26         09       $11,750
    ALBIN               79              26         10       $17,900
                              .
                              .
                              .

DTR>
```

You cannot invoke command files while you are in ADT or Guide Mode.

You can invoke a command file in response to the RW> prompt of the Report Writer. The file must begin with valid report statements. If you complete the report specification in the file with an END_REPORT statement, you can follow the specification with other valid DATATRIEVE commands or statements. When you invoke a command file, DATATRIEVE prints each command or statement on your terminal and executes it as if you had entered it directly from your keyboard. If an error occurs, DATATRIEVE prints an error message and stops executing the command file.

10.4 Aborting Command Files

To abort a command file that may contain an error, include an ABORT statement in the file. If the responses meet the abort conditions and SET ABORT is in effect, DATATRIEVE aborts the command file and prints the message specified for the ABORT command. If SET NO ABORT is in effect, DATATRIEVE aborts the command or statement that contains the ABORT but continues to execute the commands and statements that follow in the file.

10.5 Editing a Command File

To edit a command file you must exit from DATATRIEVE and use a text editor. When you correct any error, return to DATATRIEVE, ready the necessary domains, establish any appropriate collections, and execute the command file again.

10.6 Sample Command File

In contrast to the sample procedure, the sample command file prints each statement and command in the file as DATATRIEVE executes it.

When DATATRIEVE encounters the statement with the *."YES OR NO" prompting value expression, it pauses to wait for your response to the question: HAVE YOU ESTABLISHED A COLLECTION? The Boolean expression CONTAINING checks your response to the question. If the response contains a letter N anywhere, the command file aborts.

When DATATRIEVE encounters the *."OUTPUT DEVICE OR FILE" prompt, it pauses again for you to select the device or file for output of the report.

Note that, except for the report name, the report that the command file produces is the same as the one produced by the procedure YACHT_SUMMARY in Chapter 9.

The file YSUM.CMD contains the following sequence of commands and statements:

```
SET ABORT
!THIS REPORT REQUIRES AN ESTABLISHED COLLECTION,
!SORTED BY LOA AND BEAM.
!
!HAVE YOU ESTABLISHED A COLLECTION?
IF *."YES OR NO" CONTAINING "N" THEN ABORT "SORRY, NO  COLLECTION."
REPORT ON *."OUTPUT DEVICE OR FILE"
SET REPORT_NAME="SAMPLE REPORT"/"FROM A COMMAND FILE"
SET LINES_PAGE=55, COLUMNS_PAGE=60
PRINT BUILDER, MODEL, LOA, BEAM, PRICE
AT BOTTOM OF LOA PRINT SKIP, "AVERAGE PRICE =",
  AVERAGE (PRICE), SKIP
AT BOTTOM OF REPORT PRINT COL 13,"NUMBER OF BOATS = ",
  COL 33, COUNT, SKIP, "AVERAGE PRICE OF ALL BOATS =", AVERAGE (PRICE)
END_REPORT
```

When you have readied the domain and established the appropriate collection, you invoke the command file with an at sign (@). You do not have to include the .CMD extension. DATATRIEVE prints each command and statement as it executes them:

```
DTR> READY YACHTS RET
DTR> FIND FIRST 5 YACHTS WITH LOA BETWEEN 36 37 AND PRICE NE 0 RET
[5 records found]
DTR> SORT BY LOA, BEAM RET
DTR> @YSUM RET
SET ABORT
!THIS REPORT REQUIRES AN ESTABLISHED COLLECTION,
!SORTED BY LOA AND BEAM.
!
!HAVE YOU ESTABLISHED A COLLECTION?
IF *."YES OR NO" CONTAINING "N" THEN ABORT "SORRY, NO  COLLECTION."
Enter YES OR NO: YES RET
REPORT ON *."OUTPUT DEVICE OR FILE"
SET REPORT_NAME="SAMPLE REPORT"/"FROM A COMMAND FILE"
SET LINES_PAGE=55, COLUMNS_PAGE=60
PRINT BUILDER, MODEL, LOA, BEAM, PRICE
AT BOTTOM OF LOA PRINT SKIP, "AVERAGE PRICE =",
  AVERAGE (PRICE), SKIP
AT BOTTOM OF REPORT PRINT COL 17,"NUMBER OF BOATS = ",
  COL 40, COUNT, SKIP, "AVERAGE PRICE OF ALL BOATS =", AVERAGE (PRICE)
END_REPORT
Enter OUTPUT DEVICE OR FILE: TI: RET
```

```
                    SAMPLE REPORT                01-Apr-87
                  FROM A COMMAND FILE            Page 1

                                        LENGTH
                                         OVER
 MANUFACTURER          MODEL             ALL       BEAM      PRICE

  ISLANDER          36                   36         11      $31,730
  I. TRADER         37                   36         12      $39,500

AVERAGE PRICE =                                               $35,615

  IRWIN             37 MARK II           37         11      $36,950
  NORTHERN          37                   37         11      $50,000
  ALBERG            37 MK II             37         12      $36,951

AVERAGE PRICE =                                               $41,300

                NUMBER OF BOATS =                  5
                AVERAGE PRICE OF ALL BOATS =                  $39,026
```

10.7 Nesting Command Files

You can invoke both procedures and command files from within a command file. For example, the command file MSMOD creates a loop with a FOR statement and then invokes the command file MOD. MOD contains a BEGIN-END block of statements that allows you to modify prices interactively:

```
DTR> @MSMOD RET
READY YACHTS WRITE
FOR YACHTS WITH RIG = "MS"
@MOD
   BEGIN
     PRINT
     IF *."Y TO MODIFY, N TO SKIP" CONTAINING "Y"
     THEN MODIFY PRICE ELSE
     PRINT "NO CHANGE"
     IF *."Y TO CONTINUE, N TO ABORT" CONTAINING "N"
     ABORT "END OF PRICE CHANGES"
   END

                                LENGTH
                                 OVER
MANUFACTURER   MODEL     RIG     ALL    WEIGHT BEAM  PRICE

 AMERICAN     26-MS       MS       26     5,500  08  $18,950
Enter Y TO MODIFY, N TO SKIP: Y RET
Enter PRICE: 19350 RET
Enter Y TO CONTINUE, N TO ABORT: Y RET
 EASTWARD     HO          MS       24     7,000  09  $15,900
Enter Y TO MODIFY, N TO SKIP: N RET
NO CHANGE
Enter Y TO CONTINUE, N TO ABORT: N RET
ABORT: END OF PRICE CHANGES
DTR> FIND YACHTS WITH RIG = "MS" RET
[5 records found]
```

```
DTR> SELECT RET
DTR> PRINT RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 AMERICAN     26-MS       MS      26     5,500  08  $19,350

DTR>
```

When nesting command files, do not allow a command file to invoke itself, either directly or indirectly. If you do, you receive the following message:

```
Command file nesting limit exceeded
```

10.8 Using a Command File in a FOR or REPEAT Statement

You can invoke a command file in a loop you create with the FOR or REPEAT statements. As the following example shows, you must be sure to invoke the command file on a separate line, and you must include its statements within a BEGIN-END block:

```
DTR> SET NO PROMPT RET
DTR> READY YACHTS WRITE RET
DTR> FIND YACHTS WITH RIG = "KETCH" RET
[13 records found]
DTR> FOR CURRENT @MOD RET
Expected statement, encountered "@".
DTR> FOR CURRENT RET
CON> @MOD RET
BEGIN
  PRINT
  IF *."Y TO MODIFY, N TO SKIP" CONTAINING "Y"
  THEN MODIFY PRICE ELSE
  PRINT "NO CHANGE"
  IF *."Y TO CONTINUE, N TO ABORT" CONTAINING "N"
  ABORT "END OF PRICE CHANGES"
END

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 ALBERG      37 MK II   KETCH   37    20,000  12  $36,951
Enter Y TO MODIFY, N TO SKIP: ^Z
Execution terminated by operator
DTR>
```

10.9 Maintaining Command Files

Your operating system directories (not the dictionary) store the command files. If you adopt the convention of using .CMD as the extension for command files, you can display the names of the command files on your terminal by requesting a directory listing of *.CMD at the command level. You can adopt any other convention you wish and use the wildcard in the same manner:

```
READY

>DIR *.CMD; RET
 Name .Typ    Size   Prot    Date       SY:[1,37]
BLD11M.CMD       2   < 60> 24-Sep-82
CLASSE.CMD       3   < 60> 24-Sep-82
SAMPLE.CMD       4   < 60> 24-Sep-82
```

You can display the contents of a command file with the TYPE command at the system level:

```
>TYPE SAMPLE.CMD; RET
SET ABORT
!THIS REPORT REQUIRES AN ESTABLISHED COLLECTION,
!SORTED BY LOA AND BEAM.
!
!HAVE YOU ESTABLISHED A COLLECTION?
IF *."YES OR NO" CONTAINING "N" THEN ABORT "SORRY, NO COLLECTION."
REPORT ON *."OUTPUT DEVICE OR FILE"
SET REPORT_NAME="SAMPLE REPORT"/"FROM A COMMAND FILE"
SET LINES_PAGE=55, COLUMNS_PAGE=60
PRINT BUILDER, MODEL, LOA, BEAM, PRICE
AT BOTTOM OF LOA PRINT SKIP, "AVERAGE PRICE =",
        AVERAGE (PRICE), SKIP
AT BOTTOM OF REPORT PRINT COL 17, "NUMBER OF BOATS = ",
  COL 40, COUNT, SKIP, "AVERAGE PRICE OF ALL BOATS =", AVERAGE (PRICE)
END_REPORT
```

You can delete a command file from your directory with the operating system command level DELETE command.

```
>DELETE SAMPLE.CMD; RET
```

Chapter 11

Using DATATRIEVE Variables

A **variable** is a symbol whose value can change as you execute a program. You can use the letter A as a variable, for instance. The name of the variable stays the same, but its value can change during a DATATRIEVE session.

You use variables in DATATRIEVE:

- To assign values to fields in STORE and MODIFY statements
- As counters in FOR, REPEAT, and WHILE loops
- As conditional values in Boolean expressions

11.1 Declaring Variables

You declare a variable with a statement in this form:

DECLARE variable-name variable-definition

The variable name is the name you give to the variable. The variable definition consists of field definition clauses.

The following is an example of a DECLARE statement. Notice the similarity between the DECLARE statement and the definition of a field in a record:

```
DTR> DECLARE X PIC 9(7)V99 EDIT_STRING IS $$,$$$,$$$.99.
```

When you declare a variable, you can use any of the DATATRIEVE field definition clauses except OCCURS and REDEFINES. You must include at least one PIC, COMPUTED BY or USAGE clause. You can also use the QUERY_HEADER, QUERY_NAME, EDIT_STRING, VALID IF, and SIGN clauses.

11.2 Assigning Values to Variables

DATATRIEVE assigns a starting value to, or initializes, variables at the time you create them. Numeric variables are initialized to the value 0. Alphabetic or alphanumeric string variables are initialized to a blank field (spaces). You can assign an initial value of 0 to numeric fields or space to string fields, but it is not necessary to do this. Of course, you must always initialize a variable when you want the starting value to be other than the DATATRIEVE default.

In most cases, you use the assignment statement (=) to give a value to a variable. The assignment statement takes the following form:

variable-name = value-expression

Variable name is the name you gave the variable in the DECLARE statement. Value expression can be any one of the following:

- A literal
- A field name
- Another variable
- A prompting value expression
- Values from a table
- A statistical function
- An arithmetic expression
- A concatenated expression

Remember, however, to assign a value that is consistent with the definition of the variable. If you assign a value that is larger or of a different data type than you specified in the PIC or USAGE clauses, your results might not be what you intended.

You can also use the assignment statement to change the value of a variable at any time after initialization. The following example declares a variable, prints its initial value, then changes its value using two methods:

```
DTR> DECLARE X PIC 999 EDIT_STRING ZZ9. RET
DTR> PRINT X RET
 X
  0
DTR> X = 23; PRINT X RET
 X
 23
```

```
DTR> X = *."VALUE FOR X" RET
Enter VALUE FOR X: 456 RET
DTR> PRINT X (-) RET
456

DTR>
```

The minus sign (-) in the preceding example suppresses the heading, in this case the name of the variable (X), in a PRINT statement.

The following example illustrates another way of assigning values to a variable. The procedure NAME_LIST creates a variable (NEAT_NAME). The values of the variable are supplied from the PERSONNEL domain and computed by two fields (FIRST_NAME and LAST_NAME) in the record definition for that domain. The values for the variable change as the values for the COMPUTED BY fields change.

The procedure uses the variable both to restrict the print display to the two name fields in the record and to improve the appearance of each name by eliminating extra spaces between the first and last names:

```
DTR> SHOW NAME_LIST RET
PROCEDURE NAME_LIST
DECLARE NEAT_NAME COMPUTED BY FIRST_NAME||" "|LAST_NAME
                    QUERY_HEADER IS "EMPLOYEE NAMES".
READY PERSONNEL
PRINT NEAT_NAME OF FIRST 2 PERSONNEL
END_PROCEDURE
DTR> :NAME_LIST RET

    EMPLOYEE NAMES

CHARLOTTE SPIVA
FRED HOWL

DTR>
```

11.3 Local and Global Variables

You use the DECLARE statement to define both local and global variables. A variable you define with a BEGIN-END block is a local variable, and you can use it only within that block. A variable you define at DATATRIEVE command level is a global variable. It remains in your workspace until you release it or exit from DATATRIEVE. Use the assignment statement (variable = value) to set the variable equal to a particular value.

11.3.1 Global Variables

You can use a global variable to change values in every record in a domain. Suppose you want to assign to each boat in YACHTS a new price that is two-thirds of the present price. By using a COMPUTED BY clause in a global variable, you can apply a single formula to every yacht, as in the example that follows:

```
DTR> READY YACHTS MODIFY RET
DTR> DECLARE SALE_PRICE COMPUTED BY PRICE/1.5 RET
CON> EDIT_STRING IS $Z9,999.99. RET
DTR> SALE_PRICE = 0 RET
DTR> FOR FIRST 5 YACHTS PRINT BOAT, SALE_PRICE RET
                                LENGTH
                                 OVER                          SALE
MANUFACTURER    MODEL     RIG    ALL   WEIGHT BEAM  PRICE      PRICE

 ALBERG         37 MK II  KETCH  37    20,000  12   $36,951  $24,634.00
 ALBIN          79        SLOOP  26     4,200  10   $17,900  $11,933.33
 ALBIN          BALLAD    SLOOP  30     7,276  10   $27,500  $18,333.33
 ALBIN          VEGA      SLOOP  27     5,070  08   $18,600  $12,400.00
 AMERICAN       26        SLOOP  26     4,000  08    $9,895  $ 6,596.67

DTR>
```

The variable SALE_PRICE declared at DATATRIEVE command level remains in the workspace throughout the session. It changes its value whenever the value of PRICE changes. The variable remains in your workspace until you release it with the RELEASE command or declare another variable with the same name.

11.3.2 Local Variables

You define local variables with DECLARE statements entered in BEGIN-END blocks and THEN statements. The local variable has an effect only within the clause or statement in which you declare it.

In the following example the local variable declared in the inner statement supersedes one with the same name declared in an outer statement. Notice that the different value or different data type assigned to the inner variable has no effect on the value of the variable in the outer statement. Note also that neither local variable exists when DATATRIEVE finishes executing the compound statements containing them both:

```
DTR> SET NO PROMPT [RET]
DTR> BEGIN [RET]
CON>    DECLARE X PIC XXX. [RET]
CON>    X = "TOP" [RET]
CON>    PRINT X [RET]
CON>            BEGIN [RET]
CON>                    DECLARE X PIC 9.99. [RET]
CON>                    X = 1.23 [RET]
CON>                    PRINT X [RET]
CON>            END [RET]
CON>    PRINT X [RET]
CON> END [RET]

 X

TOP

 X

1.23

 X

TOP

DTR> PRINT X [RET]
Field "X" is undefined or used out of context

DTR>
```

If you declare a global variable and then use the same name for local variables, the value of the global variable is not affected by value assignments and changes made to its local counterparts.

11.4 Using Variables to Assign Values to Fields

You can use variables to assign values to fields in the USING clauses of STORE and MODIFY statements. You cannot, however, use a variable to respond to a prompt for a field value, whether the prompt is the result of the syntax of the STORE or MODIFY statement or of a prompting value expression.

In USING clauses of STORE and MODIFY statements, you can use value expressions on the right side of assignment statements to supply values for fields. In some circumstances, you can use variables in those assignments to control the uniformity of input data.

In this example, WORK is a domain you want to contain uniform names. The data file is indexed on WHO and allows duplicates:

```
DTR> SHOW WORK_REC RET
RECORD WORK_REC
 USING
01 TOP.
        03 JOB PIC X(15).
        03 RESPONSIBLE_PERSON PIC X(4)
           QUERY_NAME WHO.
;
DTR>
```

NAME_TABLE translates the varying inputs into uniform values to store in the work domain:

```
DTR> SHOW NAME_TABLE RET
TABLE NAME_TABLE
E                :       ED,
ED               :       ED,
EM               :       ED,
M                :       ED,
F                :       FRED,
FH               :       FRED,
FRED             :       FRED,
H                :       FRED,
L                :       RICK,
R                :       RICK,
RBL              :       RICK,
RICK             :       RICK,
RL               :       RICK,
ELSE "????"
END_TABLE
```

In the following STORE statement, the USING clause uses the variable PERSON with a prompting value expression for the responsible person. The table translates the value supplied to that prompt and stores the uniform results in the field WHO.

```
DTR> SET NO PROMPT RET
DTR> DECLARE PERSON PIC X(5). RET
DTR> READY WORK WRITE RET
DTR> REPEAT 3 STORE WORK USING RET
CON> BEGIN RET
CON>    JOB = *.JOB RET
CON>    PERSON = *.WHO RET
CON>    WHO = PERSON VIA NAME_TABLE RET
CON> END RET
Enter JOB: CLEANING RET
Enter WHO: E RET
Enter JOB: DRYING RET
Enter WHO: FR RET
Enter JOB: SELLING RET
Enter WHO: R RET
DTR> PRINT WORK RET
```

```
                    RESPONSIBLE
       JOB            PERSON

CLEANING              ED
DRYING                ????
SELLING               RICK

DTR>
```

11.5 Using Variables as Counters to Control Record Streams

You can use a counter to keep track of how many times DATATRIEVE performs a task. When you use a counter to control a record stream, however, it can also limit the number of times DATATRIEVE executes FOR and WHILE statements.

Suppose you want to keep a running count of the yachts you are repricing. You can define a global variable and use it as a counter. Each time DATATRIEVE prints the corresponding record, it increases A by one:

```
DTR> DECLARE A PIC 999. RET
DTR> A = 0 RET
DTR> PRINT A RET

 A

000

DTR> SET NO PROMPT RET
DTR> FOR YACHTS RET
CON> BEGIN RET
CON>     A = A + 1 RET
CON>     PRINT A, BOAT RET
CON> END

                                          LENGTH
                                           OVER
 A  MANUFACTURER    MODEL       RIG       ALL    WEIGHT BEAM  PRICE

001  ALBERG        37 MK II    KETCH     37     20,000  12   $36,951
002  ALBIN         79          SLOOP     26      4,200  10   $17,900
003  ALBIN         BALLAD      SLOOP     30      7,276  10   $27,500
004  ALBIN         VEGA        SLOOP     27      5,070  08   $18,600
005  AMERICAN      26          SLOOP     26      4,000  08    $9,895
006  AMERICAN      26-MS       MS        26      5,500  08   $18,895
007  BAYFIELD      30/32       SLOOP     32      9,500  10   $32,875
                                         .
                                         .
                                         .
113  WRIGHT        SEAWIND II  SLOOP     32     14,900  00   $34,480

DTR>
```

When you use a global variable as a counter in FOR and WHILE statements, you must initialize the variable to ensure that DATATRIEVE executes the loop the number of times you intend. If you use the same variable to control two loops, you must reinitialize the variable before DATATRIEVE executes the second loop. If you do not, DATATRIEVE may execute the loop fewer times than you intend. It may not execute the loop at all if the value of the variable is greater than the value specified in the IF-THEN-ELSE statement in the second loop.

Either at the beginning or the end of the loop, you can use an IF-THEN-ELSE statement to evaluate the variable against a set of conditions. Depending on the evaluation, DATATRIEVE will continue to loop or will execute an ABORT statement to end the loop. This example shows the use of a global variable to control a FOR statement and force an end to the loop:

```
DTR> SET NO PROMPT RET
DTR> READY YACHTS RET
DTR> DECLARE B PIC 9. RET
DTR> B = 0 RET
DTR> FOR YACHTS RET
CON>    BEGIN RET
CON>             B = B + 1 RET
CON>             PRINT B, BOAT RET
CON>             IF B = 7 THEN ABORT "END OF LOOP" RET
CON> END RET

                                    LENGTH
                                     OVER
B MANUFACTURER   MODEL     RIG      ALL    WEIGHT BEAM  PRICE

1  ALBERG       37 MK II   KETCH    37     20,000  12  $36,951
2  ALBIN        79         SLOOP    26      4,200  10  $17,900
3  ALBIN        BALLAD     SLOOP    30      7,276  10  $27,500
4  ALBIN        VEGA       SLOOP    27      5,070  08  $18,600
5  AMERICAN     26         SLOOP    26      4,000  08   $9,895
6  AMERICAN     26-MS      MS       26      5,500  08  $18,895
7  BAYFIELD     30/32      SLOOP    32      9,500  10  $32,875
ABORT: END OF LOOP
Execution terminated by "ABORT" statement
DTR>
```

You can also assign a value greater than zero to the variable and use it as a decremental counter, as in the following example:

```
DTR> DECLARE A PIC 9.
DTR> A=8
DTR> PRINT A RET

A

8
```

```
DTR> SET NO PROMPT RET
DTR> FOR YACHTS RET
CON> BEGIN RET
CON>    PRINT A, BOAT RET
CON>    A = A - 1 RET
CON>    IF A = 0 THEN ABORT "END OF LOOP" RET
CON> END RET

                                    LENGTH
                                     OVER
A MANUFACTURER   MODEL     RIG      ALL    WEIGHT BEAM  PRICE

8  ALBERG       37 MK II   KETCH    37     20,000  12   $36,951
7  ALBIN        79         SLOOP    26      4,200  10   $21,659
6  ALBIN        BALLAD     SLOOP    30      7,276  10   $27,500
5  ALBIN        VEGA       SLOOP    27      5,070  08   $18,600
4  AMERICAN     26         SLOOP    26      4,000  08    $9,895
3  AMERICAN     26-MS      MS       26      5,500  08   $18,895
2  BAYFIELD     30/32      SLOOP    32      9,500  10   $32,875
1  BLOCK I.     40         SLOOP    39     18,500  12   $29,050
ABORT: END OF LOOP
Execution terminated by "ABORT" statement
DTR>
```

The WHILE statement causes DATATRIEVE to repeat a statement as long as the condition specified in the Boolean expression is "true." The command file (WH.CMD) in the following example uses a variable in a WHILE statement:

```
DTR> @WH RET
DECLARE A USAGE INTEGER.
A = 0
PRINT A

  A

    0

FOR YACHTS
WHILE A < 5
        BEGIN
           A = A + 1
           PRINT A, BOAT
        END

                                          LENGTH
                                           OVER
  A   MANUFACTURER   MODEL      RIG       ALL    WEIGHT BEAM  PRICE

    1  ALBERG       37 MK II   KETCH     37     20,000  12   $36,951
    2  ALBERG       37 MK II   KETCH     37     20,000  12   $36,951
    3  ALBERG       37 MK II   KETCH     37     20,000  12   $36,951
    4  ALBERG       37 MK II   KETCH     37     20,000  12   $36,951
    5  ALBERG       37 MK II   KETCH     37     20,000  12   $36,951

DTR>
```

The variable in the next example changes with the value expression PRICE/LOA and stops the FOR loop when the value expression meets the condition specified in the IF-THEN-ELSE statement:

```
DTR> SET NO PROMPT RET
DTR> FIND FIRST 15 YACHTS RET
[15 records found]
DTR> DECLARE X PIC 9999 EDIT_STRING IS $$,$$$.99. RET
DTR> X = 0 RET
DTR> FOR CURRENT RET
CON>    BEGIN RET
CON>          X = PRICE/LOA RET
CON>          PRINT TYPE, X ("PRICE"/"PER FOOT") RET
CON>          IF X GE 1000 THEN ABORT "TOO SHORT FOR THE MONEY" RET
CON>    END RET

                          PRICE
MANUFACTURER   MODEL     PER FOOT

 ALBERG       37 MK II     $998.00
 ALBIN        79           $688.00
 ALBIN        BALLAD       $916.00
 ALBIN        VEGA         $688.00
 AMERICAN     26           $380.00
 AMERICAN     26-MS        $726.00
 BAYFIELD     30/32      $1,027.00

ABORT: TOO SHORT FOR THE MONEY
Execution terminated by "ABORT" statement
DTR>
```

Chapter 12

Using DATATRIEVE Tables

Tables save space. They save space in record definitions and data files. They also save space when you type in commands and statements. With a DATATRIEVE table, you can associate codes with corresponding translations—for example, products with price codes.

One advantage to such code and translation pairs is that you can substitute a short field for a long one. For instance, you can have a list of codes for job titles such that you can put E01 into the definition and have a table that translates the code into Tax Assessor First Class. A dictionary table looks like the sample in Figure 12–1.

Figure 12–1: Code and Translation Pairs in a Dictionary Table

DICTIONARY TABLE

Code	Translation
C	"Customer Services"
E	Engineering

ZK-0970A-HC

12.1 Using a Dictionary Table

Suppose, for example, that you are a manufacturer who needs to order various items from clerks around the country. With the help of a DATATRIEVE table, you can use one command to find out which products have reached

zero inventory, which clerks are responsible for the parts, and the phone numbers for those clerks.

You can use a dictionary table, in this case a table called ORDER_TABLE, to pair clerks and phone numbers with the products you need to track. You can define a procedure to determine which items are out of stock, then invoke a table producing a list of clerks responsible for those parts.

The following example uses the domain PRODUCTS and a procedure TAB, which finds the items no longer in stock and uses ORDER_TABLE to list the clerks you need to call:

```
DTR> READY PRODUCTS RET
DTR> PRINT ALL PRODUCTS RET

                                                    PARTS
                                            PART     IN
        VENDOR                  ITEM        NUMBER  STOCK

ACME ASPHALT & SHINGLE    ASPHALT             1001     3
ACME ASPHALT & SHINGLE    SHINGLES            1002     0
ACME ASPHALT & SHINGLE    CUBE WALLS          1003  9999
PURGE SYSTEMS             ERASER-CHALK        3001     3
PURGE SYSTEMS             ERASER-PENCIL       3002     1
PURGE SYSTEMS             WHITE-OUT           3003  8645
PURGE SYSTEMS             MAGNETS-20 OZ.      3004     0
QUERY ENTERPRISES         LISTINGS            2001     4
QUERY ENTERPRISES         REPORTS             2002     0

DTR> SHOW ORDER_TABLE RET
TABLE ORDER_TABLE
"ACME ASPHALT & SHINGLE"    : "L. LANDFILL  (999) 555-1234",
"QUERY ENTERPRISES"         : "T. ABMOW  (111) 555-4321",
"PURGE SYSTEMS"             : "T. SKWAIRDEE  (123) 555-9876",
ELSE "SOMETHING ELSE"
END_TABLE
DTR> SHOW TAB RET
PROCEDURE TAB
FIND PRODUCTS WITH STOCK = 0 SORTED BY PART
FOR CURRENT PRINT ITEM, PART,
VENDOR VIA ORDER_TABLE ("CALL") USING X(30)
END_PROCEDURE
DTR> :TAB RET

                  PART
        ITEM     NUMBER            CALL

SHINGLES           1002  L. LANDFILL  (999) 555-1234
REPORTS            2002  T. ABMOW  (111) 555-4321
MAGNETS-20 OZ.     3004  T. SKWAIRDEE  (123) 555-9876

DTR>
```

12.2 Creating Dictionary Tables

To create a dictionary table, enter the DEFINE TABLE command:

```
DEFINE TABLE table-name
```

Then enter the pairs of codes and their translations on separate lines, separating them by a colon (:). Make sure to place a comma after each pair at the end of the line.

If the code or the translation contains more than one word, enclose it in quotation marks. For example, the translation "New Hampshire" must be placed in quotation marks so DATATRIEVE will recognize the two words as one translation entry. You must also use quotation marks to preserve lowercase letters in a code or translation.

If you use quotation marks, the rules for character string literals apply. That is, neither the code nor the translation can exceed 132 characters, and neither can contain a RETURN, line feed, space, or control character.

After the last code and translation pair, you can enter an optional ELSE clause. DATATRIEVE substitutes the translation in the ELSE clause for any values not found in the table. If your dictionary table does not contain an ELSE clause, DATATRIEVE prints an error message when it cannot find a value in the table.

Translations included in an ELSE clause are also enclosed in quotation marks. Apply the same rules as with character string literals, except do not place a comma after an ELSE clause.

To end a dictionary table definition, enter the keyword END_TABLE. The END_TABLE statement must follow the ELSE clause, if specified, or the last code and translation pair if there is no ELSE clause. If you omit the ELSE clause, do not put a comma after the last code and translation pair.

After you enter the END_TABLE statement, DATATRIEVE stores the definition in your current dictionary and creates an access control list for the dictionary table.

When you first refer to a table, DATATRIEVE searches the current data dictionary for the table and loads it into your DATATRIEVE workspace. DATATRIEVE evaluates the value expression in your statement and compares the value with the codes in the table. The comparison is case sensitive and proceeds character-by-character. Thus, a value expression of 5 does not match a code of 05, and a value expression of Rig does not match a code of RIG.

As you define a dictionary table, DATATRIEVE checks for syntax errors. For example, if you use a semicolon in place of the required colon, DATATRIEVE prints an error message and aborts the DEFINE TABLE command.

DATATRIEVE does not store anything in the dictionary until you complete the table definition without a syntax error. You can use the EDIT command to change a table once it is in the dictionary.

If you want to edit the table as you create it, you can:

- Enter one code and translation pair, finish the definition with END_TABLE, then use the EDIT command to extend the table to include additional code and translation pairs.
- Leave DATATRIEVE and use the editor you usually use on your operating system to define the table in a command file. Then enter DATATRIEVE again and invoke the command file at the DTR> prompt. DATATRIEVE checks the syntax, and if the definition contains an error you can leave DATATRIEVE again and edit the command file.

  See Chapter 10 for more information about command files.

12.3 Sample Dictionary Tables

This section show ways you can define and use dictionary tables.

For example, suppose you are a sales manager with a list of names and phone numbers of potential customers, and you want to know where they live. You can enter telephone area codes and their corresponding names in a dictionary table, then refer to the table in a PRINT statement.

You can define a dictionary table as follows:

```
TABLE AREA_CODE_TABLE
207 : "MAINE",
603 : "NEW HAMPSHIRE",
802 : "VERMONT",
617 : "BOSTON AREA",
508 : "EAST-CENTRAL MASSACHUSETTS",
413 : "WESTERN MASSACHUSETTS",
401 : "RHODE ISLAND",
203 : "CONNECTICUT",
ELSE "NOT A VALID AREA CODE"
END_TABLE
```

Then you can prompt for an area code and print the corresponding state in a single statement:

```
DTR> PRINT *."AREA CODE" VIA AREA_CODE_TABLE USING X(21) RET
Enter AREA CODE: 203 RET
CONNECTICUT
DTR>
```

Suppose you also want to create a table of abbreviations for RIG, KETCH, SLOOP, and MS from the YACHTS domain. You can create the following dictionary table, RIG_TABLE, and store it in the data dictionary for YACHTS:

```
DTR> DEFINE TABLE RIG_TABLE RET
DFN> "SLOOP" : "ONE-MAST", RET
DFN> "KETCH" : "TWO MASTS, BIG ONE IN FRONT", RET
DFN> "YAWL"  : "SIMILAR TO KETCH", RET
DFN> "MS"    : "SAILS AND BIG MOTOR", RET
DFN> ELSE "SOMETHING ELSE" RET
DFN> END_TABLE RET
DTR>
```

12.4 Using the IN Relational Operator with DATATRIEVE Tables

You can use tables with the relational operators IN and NOT IN to set conditions in IF-THEN-ELSE statements and to validate data. The following is the general format for using IN and NOT IN with a DATATRIEVE table:

```
field-name
* prompt     [NOT] IN table-name
variable-name
```

12.4.1 Using a Table in a Record Selection Expression

In a record selection expression, you can use a Boolean expression that refers to a dictionary table:

```
DTR> FOR YACHTS WITH RIG NOT IN RIG_TABLE PRINT BOAT RET
```

```
                                    LENGTH
                                     OVER
MANUFACTURER    MODEL      RIG      ALL    WEIGHT BEAM  PRICE

 AMERICAN     26-MS       MS       26      5,500  08  $18,895
 EASTWARD     HO          MS       24      7,000  09  $15,900
 FJORD        MS 33       MS       33     14,000  11
 LINDSEY      39          MS       39     14,500  12  $35,900
 ROGGER FD    M/S         MS       35     17,600  11

DTR> FIND YACHTS WITH RIG IN RIG_TABLE RET
[108 records found]
DTR>
```

12.4.1.1 Using a Table to Set Conditions in an IF-THEN-ELSE Statement

You can combine IN with a table reference to set the conditions of an IF-THEN-ELSE statement:

```
DTR> SET NO PROMPT RET
DTR> FOR YACHTS WITH LOA = 26 RET
CON> BEGIN RET
CON>     PRINT RET
CON>     IF RIG NOT IN RIG_TABLE RET
CON>     THEN ABORT "Does not meet requirements" RET
CON> END RET

                                    LENGTH
                                     OVER
MANUFACTURER    MODEL      RIG      ALL    WEIGHT BEAM  PRICE

 ALBIN        79          SLOOP    26      4,200  10  $17,900
 AMERICAN     26          SLOOP    26      4,000  08   $9,895
 AMERICAN     26-MS       MS       26      5,500  08  $18,895
ABORT: Does not meet requirements
Execution terminated by "ABORT" statement
DTR>
```

12.4.1.2 Using a VALID IF Clause with a Table to Validate Data

By referring to a dictionary table in a VALID IF clause, you can validate data in a field before storing the data. The VALID IF clause must be part of the record definition. The following definition of PHONE_REC illustrates this method of automatic data validation:

```
DTR> DEFINE RECORD PHONE_REC USING RET
DFN> 01 PHONE. RET
DFN>     02 NAME PIC X(20). RET
DFN>     02 NUMBER PIC 9(7) EDIT_STRING IS XXX-XXXX. RET
DFN>     02 LOCATION PIC X(9). RET
DFN>     02 DEPARTMENT PIC XX VALID IF RET
DFN>             DEPARTMENT IN DEPT_TABLE. RET
DFN>; RET
DTR>
```

The table DEPT_TABLE looks like this:

```
DTR> SHOW DEPT_TABLE RET
TABLE DEPT_TABLE
"T3" : "TYPESETTING",
"D5" : "FINANCE",
"R9" : "HOUSEKEEPING",
"X7" : "GROUNDSKEEPING",
"R7" : "BEEKEEPING",
ELSE "NOT A DEPARTMENT HERE"
END_TABLE
DTR>
```

If you try to enter a department number that is not in the table, DATATRIEVE rejects the entry and prompts you for a correct one:

```
DTR> READY PHONE WRITE RET
DTR> STORE PHONE RET
Enter NAME: "BENJAMIN PAUL" RET
Enter NUMBER: 7548769 RET
Enter LOCATION: POLE_S RET
Enter DEPARTMENT: T4 RET
Validation error for DEPARTMENT
Re-enter DEPARTMENT: T3 RET
DTR>
```

12.4.2 Using the Keyword VIA with DATATRIEVE Tables

To refer to tables in value expressions, combine the table name with the keyword VIA. DATATRIEVE compares the codes in the table with the value you supply. If the value matches one of the codes, DATATRIEVE uses the corresponding translation in the table. Use the following format to refer to a dictionary table in a value expression:

field-name
*.prompt VIA table-name
variable-name

You can refer to an entry in the dictionary table RIG_TABLE by using a prompting value expression, as shown in the following PRINT statement:

```
DTR> PRINT *."TYPE OF BOAT" VIA RIG_TABLE USING X(30) RET
Enter TYPE OF BOAT: KETCH RET
TWO MASTS, BIG ONE IN FRONT
```

If you refer to a dictionary table in a PRINT statement, include an edit string to specify the number of characters to be printed. If you omit the edit string, DATATRIEVE uses an edit string that is 10 characters long.

12.5 DATATRIEVE Tables and Workspace

Once you have referred to a table, it remains in your DATATRIEVE workspace until you either relinquish it with the RELEASE command or end your DATATRIEVE session. If you are redefining a table, you must release the old version before the new table takes effect. Note that DATATRIEVE does not release tables when you switch data dictionaries.

You cannot have two tables with the same name in your workspace at the same time. Use the SHOW READY command to display the names of tables in your DATATRIEVE workspace. It is helpful for optimization to see the tables in your workspace so you can release tables you do not need.

For a complete discussion of DATATRIEVE workspace, see Chapter 17 in this manual.

12.6 Accessing Tables

This section describes how to display, edit, and delete tables using the SHOW, EDIT, and DELETE commands.

12.6.1 Listing Table Names

You can use the SHOW TABLES command to list the names of all dictionary tables in your default dictionary. For example:

```
DTR> SHOW TABLES RET

Tables:
        DEPT_TABLE      JOB_TITLE      NAME_TABLE      RIG_TABLE
DTR>
```

12.6.2 Displaying Tables

To display a dictionary table on your terminal, use the SHOW command and specify the name of the table. Use the following format to display a table:

SHOW table-name [(passwd) | (*)]

For example:

```
DTR> SHOW AREA_CODE_TABLE RET
TABLE AREA_CODE_TABLE

207 : "MAINE",
603 : "NEW HAMPSHIRE",
802 : "VERMONT",
617 : "BOSTON AREA",
508 : "EAST-CENTRAL MASSACHUSETTS",
413 : "WESTERN MASSACHUSETTS",
401 : "RHODE ISLAND",
203 : "CONNECTICUT",

ELSE "NOT A VALID AREA CODE"
END_TABLE
DTR>
```

12.6.3 Editing Tables

You can modify a table in the current data dictionary with EDT. Type EDIT and the table name, then use EDT to make the desired changes. Remember to end a new table entry with a comma. When you EXIT from EDT, DATATRIEVE places the modified table in your workspace.

See Chapter 16 for more information on EDT.

The following example illustrates how to use EDT to add a new entry to an existing table:

```
DTR> SHOW NUM_LIST RET
TABLE NUM_LIST
1:"ONE",
2:"TWO",
3:"THREE",
ELSE "NUMBER OUT OF TABLE RANGE"
END_TABLE
DTR> EDIT NUM_LIST RET
* "ELSE" RET
        ELSE "NUMBER OUT OF TABLE RANGE"
* CH RET
```

At this point, you should see the entire text of the file, insofar as it will fit on the display screen, with the cursor indicating the point where insertion will occur. The previous example will show the cursor positioned after the word ELSE. To add information before that word, use the arrow keys or keypad to position the cursor. Then type the information that you wish to add. Press CTRL/Z to return to EDT line command mode. For example:

```
4:"FOUR", RET
```

The text should then appear as follows:

```
1:"ONE",
2:"TWO",
3:"THREE",
4:"FOUR",
ELSE "NUMBER OUT OF TABLE RANGE"
END_TABLE
```

You may need to add some spaces to correct the indentation. When you press CTRL/Z to exit EDT change mode, the line command prompt (*) will reappear. Then you can type EXIT to return to the DTR> prompt.

```
*exit
DTR>
```

12.6.4 Deleting Tables

You can delete a table from your current data dictionary with the DELETE command. Use the following format:

DELETE table-name [(passwd) | (*)]

The following example deletes AREA_CODE_TABLE from the default data dictionary:

```
DTR> SHOW TABLES RET
Tables:
        AREA_CODE_TABLE JOB_TITLE       NAME_TABLE
DTR> DELETE AREA_CODE_TABLE; RET
DTR> SHOW TABLES RET
Tables:
        JOB_TITLE       NAME_TABLE
DTR>
```

12.7 Protecting Dictionary Tables

When you create a dictionary table, DATATRIEVE stores its definition in the current data dictionary and creates an access control list (ACL) for it. DATATRIEVE automatically stores one project-programmer number (PPN) or user identification code (UIC) in the ACL with full access privileges. An account number is called a PPN under RSTS/E systems and a UIC under other operating systems.

Your individual installation determines the actual PPN/UIC that is stored in the ACL. Any user logged in under that PPN/UIC and the creator of the table have R (Read), W (Write), E (Execute), M (Modify), and C (Control) access to the dictionary table. Depending on the PPN/UIC in the ACL other users in the installation may be able to delete, modify, or print the dictionary table.

If you want to grant additional privileges to other users or further restrict the use of the dictionary table, you must modify its ACL. For more information on protecting dictionary tables and on modifying access control lists, refer to Chapter 19.

To guard against accidental deletion of your dictionary table, you can maintain a backup copy of it by using the DATATRIEVE EXTRACT command to copy your table to a disk file or tape file. For example:

```
DTR> EXTRACT ON NAMTAB.BKP NAME_TABLE RET
DTR>
```

Chapter 13

Defining and Using Views

A **view** is a special type of domain that lets you select some or all fields in some (or all) records from one or more domains. Using a view, you can refer to fields and field values in different domains without duplicating their records and data. You can use views to do the following:

- Work with subsets of records
- Refer to subsets of fields in records
- Change the apparent order of the fields in records
- Combine subsets of records from more than one domain
- Modify the values in the fields you select

A view does not actually change the way fields are organized in records or the way records are combined into data files. A view only changes how existing data appears to you. You can modify values in the fields selected by the view, but you cannot store or erase any records accessed by the view.

A view is like a window in a house. Just as a window might let you look into more than one room, a view can let you look into more than one domain. A window might give you a complete look at everything in one or more rooms, restrict what you can see to a few items of furniture, or only let you see the back of one chair. Similarly, a view can let you look at all records, a few records, or parts of records.

Each window in a house gives you a different picture of what is inside, but the actual locations of the items you see are the same no matter how you look at them. In the same way, when you define a view, you do not affect the way information is actually stored.

13.1 Defining Views

You define a view by creating a domain definition in your data dictionary with the DEFINE DOMAIN command:

```
DEFINE DOMAIN view-name OF domain-name [,...]   USING
        level-number-1 field-name-1     OCCURS FOR rse-1 .

        level-number-2 field-name-2   { OCCURS FOR rse-2   }.
                                      { FROM domain-name-2 }

              .              .               .
              .              .               .
              .              .               .
;
```

After the keyword OF, you must list each domain that the view uses. The domains you list cannot be views themselves. You may specify the domains in any order, separating them with commas. You must end each field definition with a period and end the view definition with a semicolon.

You can use two clauses to define the fields in a view:

- OCCURS FOR
- FROM

The top-level field must be defined with an OCCURS FOR clause. The record selection expression in the first OCCURS FOR clause determines the number of records in the view. Each subsequent OCCURS FOR clause creates a list within the view. Consequently, a view that contains more than one OCCURS FOR clause is always a hierarchy. (The first OCCURS FOR clause does not make the view a hierarchy. It only establishes the source record stream for the view.)

See Chapter 14 for more information about hierarchies.

You establish the fields of data for the view with the FROM clause. The FROM clause specifies the name of the field and the domain from which it is derived. The domain must be the same domain named in the preceding OCCURS FOR clause. The field name must be either a field name or a query name from that domain.

If two or more fields have the same name, you might have to qualify those field names to avoid ambiguity. See Appendix A for more information about qualified field names.

13.1.1 Views Using Subsets of Records

A view lets you work with a specific subset of records from another domain. For instance, you may want to work with the records for ketches only and no other rig type. The following example shows a view definition that allows you to work with four fields of the yachts that are ketches:

```
DTR> DEFINE DOMAIN KETCHES RET
DFN> OF YACHTS USING RET
DFN> 01 KETCH OCCURS FOR YACHTS WITH RIG EQ "KETCH". RET
DFN>    03 TYPE FROM YACHTS. RET
DFN>    03 LOA FROM YACHTS. RET
DFN>    03 PRICE FROM YACHTS. RET
DFN> ; RET
DTR> READY KETCHES RET
DTR> PRINT FIRST 4 KETCHES RET

                          LENGTH
                           OVER
MANUFACTURER    MODEL      ALL     PRICE

 ALBERG       37 MK II     37     $36,951
 CHALLENGER   41            41     $51,228
 FISHER       30            30
 FISHER       37            37

DTR>
```

The view domain KETCHES, which is based on the single domain YACHTS, is not hierarchical because there is only one OCCURS FOR clause.

You cannot store or erase records in a view. Otherwise, you can use a view just as you would any other domain.

13.1.2 Views Using Subsets of Fields

Another type of view lets you refer to a subset of fields from the records of another domain. For example, the record definition for YACHTS contains seven elementary fields and three group fields:

```
DTR> READY YACHTS RET
DTR> SHOW FIELDS RET
YACHTS
    BOAT
        TYPE [Indexed field]
            MANUFACTURER (BUILDER) [Character string, indexed key]
            MODEL [Character string, indexed key]
        SPECIFICATIONS (SPECS)
            RIG [Character string]
            LENGTH_OVER_ALL (LOA) [Character string]
            DISPLACEMENT (DISP) [Number]
            BEAM [Number]
            PRICE [Number]
```

If you want to work with only a few fields of the record, you can create a record definition for those fields and then create a domain and a data file containing one record for each record in YACHTS. The result is a data file that duplicates some field values in an existing data file (YACHT.DAT). Maintaining these two files so that they always contain the same field values would be difficult.

You can define a view, however, that lets you look at just the fields in YACHTS that you need without duplicating field values. You also avoid the additional time and overhead of creating another record definition and creating and updating two data files:

```
DTR> DEFINE DOMAIN MAKERS RET
DFN> OF YACHTS USING RET
DFN> 01 BOAT OCCURS FOR YACHTS. RET
DFN>    03 TYPE FROM YACHTS. RET
DFN>    03 RIG FROM YACHTS. RET
DFN> ;
DTR> READY MAKERS RET
DTR> PRINT FIRST 6 MAKERS RET

MANUFACTURER    MODEL      RIG

 ALBERG         37 MK II   KETCH
 ALBIN          79         SLOOP
 ALBIN          BALLAD     SLOOP
 ALBIN          VEGA       SLOOP
 AMERICAN       26         SLOOP
 AMERICAN       26-MS      MS
```

13.1.3 Views Using More than One Domain

The preceding sections described how to use view domains to define a subset of records or fields from a single domain. You can also use field values from more than one domain.

The domain OWNERS, for example, contains records of yacht owners. Each record contains the owner's name and the name, builder, and model of the owner's yacht:

```
DTR> SHOW OWNER_RECORD RET
RECORD OWNER_RECORD
ALLOCATION IS LEFT_RIGHT
01 OWNER.
   03 NAME PIC X(10) QUERY_HEADER IS "OWNER"/"NAME"
      EDIT_STRING IS X(5).
   03 BOAT_NAME PIC X(17) QUERY_HEADER IS "BOAT NAME".
   03 TYPE.
      06 BUILDER PIC X(10).
      06 MODEL PIC X(10).
;
DTR> READY OWNERS RET
DTR> PRINT FIRST 1 OWNERS RET

OWNER
NAME       BOAT NAME       BUILDER      MODEL

SHERM MILLENNIUM FALCON ALBERG      35

DTR>
```

If you define a view that refers to both the OWNERS domain and the YACHTS domain, you can use any combination of fields and records in those domains. For example, you can include the name and yacht fields from the OWNERS domain and the price field from the YACHTS domain.

The view SAILBOATS in the following example uses every field and every record in the YACHTS domain to match owners with the boats they own. It uses records from the OWNERS domain and matches those records by boat type with records in the YACHTS domain. It requests only the NAME field from those records.

After the first OCCURS FOR clause, each OCCURS FOR creates a list within the view. A domain that contains a list is a hierarchy. The view SAILBOATS, therefore, is a hierarchical view:

```
DTR> SHOW SAILBOATS RET
DOMAIN SAILBOATS
 OF YACHTS, OWNERS USING
01 SAILBOAT OCCURS FOR YACHTS.
   03 BOAT FROM YACHTS.
   03 SKIPPERS OCCURS FOR OWNERS WITH TYPE EQ BOAT.TYPE.
      05 NAME FROM OWNERS.
;
DTR>
```

The field SKIPPERS is a list of owner names. Each record in SAILBOATS can contain a NAME field for each of several owners. This view allows you to combine all the information on each yacht with the names of all its skippers.

```
DTR> READY SAILBOATS RET
DTR> PRINT FIRST 2 SAILBOATS WITH ANY SKIPPERS RET

                                  LENGTH
                                   OVER                                  OWNER
MANUFACTURER     MODEL      RIG    ALL    WEIGHT   BEAM     PRICE       NAME

ALBIN            VEGA       KETCH  35     17,000   12       $33,000     STEVE
                                                                        HUGH
C & C            CORVETTE   SLOOP  31      8,650   01                   JIM
                                                                        ANN
DTR>
```

See Section 13.2.1 for further examples of printing list field values. Hierarchies are discussed in greater detail in Chapter 14.

In general, if you define a view that uses only one domain, use an OCCURS FOR clause to define the top-level field and then use FROM clauses to specify which fields the view contains. If you define a view using more than one domain, group the field definitions by the domain they refer to, putting the field definition with an OCCURS FOR clause first in each group.

13.2 Using a View Domain

You use a view as you do any other domain. To ready a view, you must have R (Read) access privilege to the view, and you must also have the same access privilege to each domain the view uses. You must have E (Execute) access privilege to the record definition associated with each domain.

You cannot store or erase records in a view, but you can modify values of fields. For example, here is how to modify a field in the view KETCHES:

```
DTR> READY KETCHES RET
DTR> SHOW KETCHES RET
DOMAIN KETCHES
 OF YACHTS USING
01 KETCH OCCURS FOR YACHTS WITH RIG EQ "KETCH".
  03 TYPE FROM YACHTS.
  03 LOA FROM YACHTS.
  03 PRICE FROM YACHTS.
;

DTR> READY KETCHES MODIFY RET
DTR> FIND KETCHES WITH PRICE EQ 0 RET
[4 records found]
DTR> PRINT ALL RET
```

```
                          LENGTH
                           OVER
MANUFACTURER    MODEL      ALL     PRICE

 FISHER         30          30
 FISHER         37          37
 PEARSON        365         36
 PEARSON        419         42

DTR> FOR CURRENT PRINT THEN MODIFY PRICE RET

                          LENGTH
                           OVER
MANUFACTURER    MODEL      ALL     PRICE

 FISHER         30          30
Enter PRICE: 30000 RET
 FISHER         37          37
Enter PRICE: 45000 RET
 PEARSON        365         36
Enter PRICE: 32000 RET
 PEARSON        419         42
Enter PRICE: 54000 RET

DTR> PRINT ALL

                          LENGTH
                           OVER
MANUFACTURER    MODEL      ALL     PRICE

 FISHER         30          30     $30,000
 FISHER         37          37     $45,000
 PEARSON        365         36     $32,000
 PEARSON        419         42     $54,000

DTR>
```

13.2.1 Using a View that Contains a List

To refer to field values contained in a list, use one of the methods described in Chapter 14. The following example uses the FIND and SELECT method described in Section 14.1.

```
DTR> SHOW SAILBOATS RET
DOMAIN SAILBOATS
 OF YACHTS, OWNERS USING
01 SAILBOAT OCCURS FOR YACHTS.
   03 BOAT FROM YACHTS.
   03 SKIPPERS OCCURS FOR OWNERS WITH TYPE EQ BOAT.TYPE.
      05 NAME FROM OWNERS.
;
DTR> READY SAILBOATS WRITE RET
DTR> FIND OWNED IN SAILBOATS WITH ANY SKIPPERS RET
[6 records found]

DTR> PRINT ALL RET
```

```
                                LENGTH
                                 OVER                                  OWNER
MANUFACTURER    MODEL     RIG    ALL    WEIGHT BEAM   PRICE   OWNER NAME
 ALBIN          VEGA      SLOOP  27      5,070  08   $18,600 STEVE
                                                              HUGH
 C & C          CORVETTE  SLOOP  31      8,650  09           JIM
                                                              ANN
 ISLANDER       BAHAMA    SLOOP  24      4,200  08    $6,500 JIM
                                                              ANN
                                                              STEVE
                                                              HARVE
 PEARSON        10M       SLOOP  33     12,441  11           TOM
 PEARSON        26        SLOOP  26      5,400  08           DICK
 RHODES         SWIFTSURE SLOOP  33     14,000  10           JOHN

DTR> SELECT 3 RET
DTR> FIND SKIPPERS RET
[4 records found]
DTR> PRINT ALL RET

OWNER
NAME

JIM
ANN
STEVE
HARVE

DTR> SELECT 2 RET
DTR> MODIFY NAME RET
Enter NAME: ANNE RET
DTR> PRINT BOAT, ALL SKIPPERS SORTED BY NAME OF OWNED RET

                                LENGTH
                                 OVER                                  OWNER
MANUFACTURER    MODEL     RIG    ALL    WEIGHT BEAM   PRICE   OWNER NAME
 ALBIN          VEGA      SLOOP  27      5,070  08   $18,600 HUGH
                                                              STEVE
 C & C          CORVETTE  SLOOP  31      8,650  09           ANN
                                                              JIM
 ISLANDER       BAHAMA    SLOOP  24      4,200  08    $6,500 ANNE
                                                              HARVE
                                                              JIM
                                                              STEVE
 PEARSON        10M       SLOOP  33     12,441  11           TOM
 PEARSON        26        SLOOP  26      5,400  08           DICK
 RHODES         SWIFTSURE SLOOP  33     14,000  10           JOHN

DTR>
```

Chapter 14

Using Hierarchies

In DATATRIEVE, the term **hierarchy** refers to a one-to-many relationship between record sources.

With hierarchies, you can nest record streams to see a single record from one record source displayed with a combination of records from another record source. This nesting established a *parent-child* relationship between the two record streams. For each record in the outer, *parent* record stream, you see all records in the inner, *child* record stream. Parent records are displayed even if there are no corresponding child records in the inner record stream. Some examples of how this can be useful are:

- One team with several players
- One project with several workers
- One employee with several previous jobs
- One library with many books
- One computer with several users

Hierarchies let you define records with fields that are lists. Items in a list can contain more than one field, and the list itself may contain more than one item. Therefore, a list lets you store more than one value for a field or group of fields in one record. Lists are also called repeating fields.

When you retrieve a value from a record containing a repeating field, you cannot always apply the same statements you do for other records. The following sequence of statements shows what can happen when you try to print the repeating field KIDS from the hierarchical record FAMILIES:

```
DTR> READY FAMILIES RET
DTR> SHOW FAMILY_REC RET
RECORD FAMILY_REC
01 FAMILY.
   03 PARENTS.
      06 FATHER PIC X(10).
      06 MOTHER PIC X(10).
   03 NUMBER_KIDS PIC 99 EDIT_STRING IS Z9.
   03 KIDS OCCURS 0 TO 10 TIMES DEPENDING ON NUMBER_KIDS.
      06 EACH_KID.
         09 KID_NAME PIC X(10) QUERY_NAME IS KID.
         09 AGE PIC 99 EDIT_STRING IS Z9.
;

DTR> PRINT FATHER OF FAMILIES RET

  FATHER

JIM
JIM
 .
 .
 .

DTR> PRINT MOTHER OF FAMILIES RET

  MOTHER

ANN
LOUISE
 .
 .
 .

DTR> PRINT KIDS OF FAMILIES RET
Expected end of statement, encountered "OF"
DTR>
```

You can print the names of fathers and mothers, but you get an error message when you try to print the list field KIDS. If you form a collection, you can again print information on fathers and mothers but not kids:

```
DTR> FIND FAMILIES RET
[13 records found]
DTR> PRINT ALL FATHER RET

  FATHER

JIM
JIM
 .
 .
 .

DTR> PRINT ALL MOTHER RET
```

```
  MOTHER

ANN
LOUISE
 .
 .
 .
DTR> PRINT ALL EACH_KID RET
Field "EACH KID" is undefined or used out of context
DTR> PRINT ALL KIDS RET
Field "KIDS" is undefined or used out of context
DTR> PRINT ALL KIDS OF FAMILIES RET
Expected end of statement, encountered "OF"
```

In the first two examples, you get a message stating that the field name is undefined or used out of context. The third example results in the same message you got in the previous example. To retrieve the information, you can apply one of the following methods to set up a DATATRIEVE context:

- Use a FIND statement to establish a context for the list. Then use a SELECT statement to identify one record in the collection.
- Use nested FOR rse loops. The outer FOR loop forms a target stream of hierarchical records and the inner FOR loop forms a stream of list items within a hierarchical record.
- Use inner print lists (ALL print-list OF rse) to form a stream of list items within a record stream.

The following sections describe these methods for retrieving items from lists. For more information about DATATRIEVE context, see Appendix A in this manual.

14.1 Retrieving Repeating Field Values with FIND and SELECT Statements

You use the FIND statement to find all the records in the file that meet your specifications. Then you can use the SELECT statement to request any one of these records:

```
DTR> READY FAMILIES RET
DTR> FIND FAMILIES RET
[14 records found]
DTR> SELECT 3; PRINT RET

                        NUMBER     KID
  FATHER      MOTHER     KIDS      NAME     AGE

JOHN         JULIE          2    ANN         29
                                 JEAN        26
```

When you have selected a record that contains a list, you can treat the list as though it were a source of records like a domain or collection. You can continue as follows:

```
DTR> PRINT KIDS RET

   KID
   NAME     AGE

ANN         29
JEAN        26

DTR>
```

You can also combine the FIND and SELECT statements to single out one list item. Then the context of the selected list item allows you to use the list item name by itself in a PRINT statement. Continue the previous example by forming a collection of the KIDS list field and selecting a list item from the collection:

```
DTR> FIND KIDS RET
[2 records found]
DTR> SELECT 2; PRINT RET

   KID
   NAME     AGE

JEAN        26

DTR> PRINT AGE RET

AGE

26

DTR>
```

14.2 Retrieving Repeating Field Values with Nested FOR Loops

To retrieve values from list items by nesting FOR loops, start from the top of the hierarchy and work toward the list items you want to retrieve. In the following example, the source for the RSE in the first or outer FOR loop is the hierarchical domain FAMILIES. The source in the second loop is the list item KIDS:

```
DTR> FOR FAMILIES RET
[Looking for statement]
CON> FOR KIDS WITH AGE < 10 RET
[Looking for statement]
CON> PRINT KID_NAME RET
```

```
   KID
   NAME

URSULA
RALPH
CHRISTOPHR
SCOTT
BRIAN
DAVID
PATRICK
SUZIE

DTR>
```

The FOR statement preceding the PRINT statement in the following example loops through all the records in FAMILIES. For each of those records, the RSE in the PRINT statement retrieves only the first kid whose age is less than 10:

```
DTR> FOR FAMILIES RET
[Looking for statement]
CON> PRINT KID_NAME OF FIRST 1 KIDS WITH AGE < 10 RET

   KID
   NAME

URSULA
CHRISTOPHR
SCOTT
DAVID
PATRICK

DTR>
```

The OF rse clause in the PRINT statement serves the same purpose as a nested FOR rse statement. The inner RSE (FIRST 1 KIDS WITH AGE < 10) identifies items from the list field KIDS that are included with a FAMILIES record identified by the outer FOR rse statement.

You get the same results using the following nested FOR rse statements:

```
DTR> FOR FAMILIES FOR FIRST 1 KIDS WITH AGE < 10 PRINT KID_NAME
```

14.3 Retrieving Repeating Field Values with Inner Print Lists

The simplest way to print a repeating field is to print the entire record containing the repeating field:

```
DTR> READY FAMILIES RET
DTR> PRINT FIRST 1 FAMILIES RET
```

```
                        NUMBER    KID
  FATHER       MOTHER    KIDS     NAME     AGE

JIM            ANN                  2    URSULA      7
                                         RALPH       3

DTR>
```

To print selected fields from the record, you must specify a print list in the PRINT statement. (Print lists consist of field names or other value expressions and modifiers.) To specify a list item in a print list, you must use an *inner print list*, which has the format:

ALL print-list OF rse

In the print list clause of the inner print list, include the list items you want to display. The OF rse of the inner print list creates a *context* for the item in the hierarchical list.

You can nest an inner print list in a PRINT statement using any of the following formats. The arrows under each format show where the inner print list begins and ends. A sample PRINT statement for the FAMILIES domain illustrates each format:

1. PRINT ALL ALL print-list OF rse OF rse

```
DTR> PRINT ALL ALL KID_NAME OF KIDS OF FIRST 1 FAMILIES RET

   KID
   NAME

URSULA
RALPH

DTR>
```

2. PRINT [ALL] value-exp, ALL print-list OF rse OF rse

```
DTR> PRINT ALL MOTHER, FATHER, ALL KID_NAME OF RET
[Looking for "FIRST", domain name, or collection name]
CON> FIRST 1 KIDS OF FIRST 1 FAMILIES RET

                            KID
  MOTHER       FATHER       NAME

ANN          JIM          URSULA

DTR>
```

3. PRINT ALL ALL print-list OF rse, value-exp OF rse

```
DTR> PRINT ALL ALL KID_NAME OF FIRST 1 KIDS, FATHER OF RET
[Looking for "FIRST", domain name, or collection name]
CON> FIRST 1 FAMILIES RET

   KID
   NAME        FATHER

URSULA       JIM

DTR>
```

There are two important points to remember when working with inner print lists:

- To DATATRIEVE, an inner print list is just another print list element in the outer print list.
- An inner print list establishes context for items in a list. See Appendix A for more information about context.

While inner print lists can complicate statements, they allow you to control completely how DATATRIEVE displays repeating fields. By using the repeating field as the source for an RSE in an inner print list, you can specify which occurrences of the repeating field DATATRIEVE displays.

As a rule of thumb, you can think of embedding an inner print list in a PRINT statement the same way you think of nesting parentheses in an arithmetic expression. Just as you put a matching left parenthesis for every right parenthesis in an arithmetic expression, there must be a matching ALL for every RSE in a PRINT statement. In the second format the first ALL is optional, but it is easier to remember the rule of thumb than to remember when ALL is optional.

The remainder of this section presents more examples to help you use inner print lists effectively.

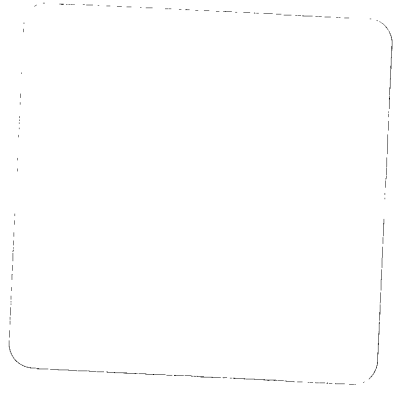
By limiting the RSEs in a PRINT statement, you can tailor a record stream to suit your needs. These next three examples show the results of limiting one or both of the RSEs in a PRINT statement that includes an inner print list using the second format:

```
DTR> PRINT ALL MOTHER, ALL EACH_KID OF KIDS OF FIRST 1 FAMILIES RET

              KID
  MOTHER      NAME     AGE

ANN         URSULA      7
            RALPH       3

DTR> PRINT ALL MOTHER, ALL EACH_KID OF FIRST 1 KIDS OF FAMILIES RET
```

```
                KID
  MOTHER        NAME      AGE

ANN             URSULA      7
LOUISE          ANNE       31
JULIE           ANN        29
ELLEN           CHRISTOPHR  0
ANNE            SCOTT       2
SARAH           DAVID       0
ANNE            PATRICK     4
MERIDETH        BEAU       28
DIDI
RUTH            ERIC       32
BETTY           MARTHA     30
LOIS            JEFF       23
SARAH           CHARLIE    31
TRINITA         ERIC       16

DTR> PRINT ALL MOTHER, ALL EACH_KID OF RET
CON> FIRST 1 KIDS OF FIRST 1 FAMILIES RET

                KID
  MOTHER        NAME      AGE

ANN             URSULA      7

DTR>
```

If you want to display only items from the list field, you must precede the inner print list with two ALL keywords, one for each RSE you define. In the following examples, the first ALL matches the RSE, FIRST 1 FAMILIES WITH NUMBER_KIDS = 3. The second ALL matches the RSE, KIDS:

```
DTR> PRINT ALL ALL EACH_KID OF KIDS OF RET
[Looking for "FIRST", domain name, or collection name]
CON> FIRST 1 FAMILIES WITH NUMBER_KIDS = 3 RET

   KID
   NAME     AGE

JEFF        23
FRED        26
LAURA       21

DTR> PRINT ALL ALL KID_NAME OF KIDS WITH RET
[Looking for Boolean expression]
CON> AGE GT 25 OF FAMILIES RET

   KID
   NAME


ANNE
JIM
ELLEN
ANN
JEAN
```



```
BEAU
BROOKS

ERIC
MARTHA
TOM
FRED
CHARLIE
HAROLD
SARAH

DTR>
```

The last display contains blank lines that you would probably want to eliminate. Eliminating empty print lines is discussed in the next section.

Note that when you include a list field in a print list, you should specify the list field as the last item. The following example shows what happens when you specify some other field after the list. The display of that field begins on the same line as the last item in the list:

```
DTR> PRINT ALL ALL EACH_KID OF KIDS, MOTHER OF FIRST 2 FAMILIES RET

   KID
   NAME     AGE    MOTHER

URSULA       7
RALPH        3   ANN
ANNE        31
JIM         29
ELLEN       26
DAVID       24
ROBERT      16   LOUISE

DTR>
```

14.4 Retrieving List Items with Nested RSEs—Eliminating Empty Print Lines

You can use Boolean expressions in the outer RSE to eliminate empty print lines. Empty print lines occur when the outer RSE includes records that do not satisfy the inner RSE. When DATATRIEVE processes a record that does not contain information needed by the inner RSE, it generates a carriage return and line feed. The following example illustrates empty print lines that result from an outer RSE (FAMILIES) that forces DATATRIEVE to include records that do not match inner RSE requirements:

```
DTR> PRINT ALL ALL EACH_KID OF KIDS WITH RET
[Looking for Boolean expression]
CON> AGE GT 25 OF FAMILIES RET
```

```
   KID
   NAME     AGE

ANNE        31
JIM         29
ELLEN       26
ANN         29
JEAN        26


BEAU        28
BROOKS      26

ERIC        32
MARTHA      30
TOM         27
FRED        26
CHARLIE     31
HAROLD      35
SARAH       27

DTR>
```

In the following statement, the clause WITH ANY KIDS eliminates the blank line caused by the record of the family without children:

```
DTR> PRINT ALL ALL EACH_KID OF KIDS WITH RET
[Looking for Boolean expression]
CON> AGE GT 25 OF FAMILIES WITH ANY KIDS RET

   KID
   NAME     AGE

ANNE        31
JIM         29
ELLEN       26
ANN         29
JEAN        26


BEAU        28
BROOKS      26
ERIC        32
MARTHA      30
TOM         27
FRED        26
CHARLIE     31
HAROLD      35
SARAH       27

DTR>
```

The four blank lines in the preceding print display represent the records of families who have kids, but whose kids have ages less than or equal to 25. The following statement excludes those records from the record stream and therefore eliminates the blank lines:

```
DTR> PRINT ALL ALL EACH_KID OF KIDS WITH RET
[Looking for Boolean expression]
CON> AGE GT 25 OF FAMILIES WITH ANY KIDS WITH RET
[Looking for Boolean expression]
CON> AGE GT 25 RET

   KID
   NAME     AGE

ANNE        31
JIM         29
ELLEN       26
ANN         29
JEAN        26
BEAU        28
BROOKS      26
ERIC        32
MARTHA      30
TOM         27
FRED        26
CHARLIE     31
HAROLD      35
SARAH       27

DTR>
```

You do not have to use an extra ALL before inner print lists if you put the PRINT statement in a FOR statement:

```
DTR> FOR FAMILIES PRINT ALL EACH_KID OF KIDS WITH RET
[Looking for Boolean expression]
CON> KID_NAME CONT "Y" RET

   KID
   NAME     AGE

JAY         22
CISSY       24
NANCY       22

DTR>
```

14.5 Retrieving Values from Sublists

You can use collections, nested FOR loops, or inner print lists to retrieve values from sublists (lists within lists).

When you work with sublists, you must remember to create a context for each level of the hierarchy. The outermost context establishes the target record or target record stream (source = FAMILIES); the second establishes the context for the list (source = KIDS); and the third establishes the context for the sublist (source = PET); and so on.

This series of FIND and SELECT statements uses collections to retrieve one list item from PET:

```
DTR> READY PETS RET
DTR> FIND PETS RET
[3 records found]
DTR> SELECT; FIND KIDS RET
[2 records found]
DTR> SELECT; FIND PET RET
[2 records found]
DTR> SELECT; PRINT PET_NAME, PET_AGE RET

   PET      PET
   NAME     AGE

POP         03

DTR>
```

You can also use nested FOR loops for dealing with a sublist:

```
DTR> FOR PETS WITH ANY KIDS RET
[Looking for statement]
CON> FOR KIDS WITH ANY PET RET
[Looking for statement]
CON> FOR PET WITH PET_AGE GT 0 RET
[Looking for statement]
CON> PRINT PET_NAME, PET_AGE RET

   PET      PET
   NAME     AGE

POP         03
SODA        04
MOUSE       03
SHORTY      08
SQUEEKY     03
FRANK       07
FRANK       14

DTR>
```

You can also use inner print lists to get at all three levels of the hierarchy:

```
DTR> PRINT ALL ALL ALL PET_NAME OF PET OF RET
[Looking for "FIRST", domain name, or collection name]
CON> KIDS WITH ANY PET WITH PET_AGE NE 0 OF RET
[Looking for "FIRST", domain name, or collection name]
CON> PETS WITH ANY KIDS WITH ANY PET WITH PET_AGE NE 0 RET
```

```
   PET
   NAME

POP
SODA
MOUSE
SHORTY
SQUEEKY
FRANK
FRANK

DTR>
```

Chapter 15

Restructuring Domains

This chapter describes how to create new domains with data from existing ones. You might do this to:

- Add new fields to the record definition associated with the domain
- Change field definitions to affect the values stored in the data file
- Create a copy of a domain for testing
- Change the file organization
- Change the index structure (key fields)
- Create a domain that contains a subset of records contained in another domain

How you create the new domain depends on whether you want to keep the old domain. If you want to keep the old domain, follow these steps when creating the new domain:

1. Define a new domain, its record, and its data file.
2. Ready the new domain for WRITE or EXTEND access and the old domain for READ access.
3. Use a statement with the following format to transfer field values from the old data file to the new one:

```
FOR rse-FROM-old-domain-name
    STORE new-domain-name USING
        new-record-name = old-record-name
```

If you want to use old procedures on the new domain, you must edit them if they refer to fields not included in the new domain. See Chapter 9 for information about editing procedures.

If the old procedures refer only to fields included in the new domain, you need not change the procedures. You can ready the new domain with the old domain name as an alias (READY NEW AS OLD) and execute the old procedures.

If you do not want to keep the old domain, you can still use the old procedures if you follow these steps:

1. Define the new domain (NEW), record (NEW_REC), and file (NEW.DAT).
2. Use a statement with the following format to transfer the data from the old domain (OLD) to the new one (NEW):

```
FOR rse-FROM-old-domain-name
    STORE new-domain-name USING
        new-record-name = old-record-name
```

3. Use the REDEFINE command, which deletes the old domain and creates a new domain with same name as the old domain. The new domain uses the old domain name (OLD), the new record definition (NEW_REC), and the new data file (NEW.DAT):

```
DTR> REDEFINE DOMAIN OLD USING NEW_REC ON NEW.DAT; RET
DTR>
```

4. Check the old procedures for any references to field names not included in the new record definition and edit where necessary.

The following sections provide examples to help you restructure your own domains.

15.1 A Sample Domain

PROJECTS is a sample domain you can create to practice restructuring:

```
DTR> SHOW PROJECTS, PROJECTS_REC RET
DOMAIN PROJECTS
 USING PROJECTS_REC ON PROJEC.DAT;
RECORD PROJECTS_REC
 USING
01 PROJECT.
   03  PROJ_CODE        PIC 9(3) QUERY_NAME IS CODE.
   03  PROJ_NAME        PIC X(10) QUERY_NAME IS NAME.
   03  MANAGER_NUM      PIC 9(5) QUERY_NAME IS NUM.
;
DTR>
```

The data file PROJEC.DAT is a sequential file and contains these records:

```
DTR> PRINT PROJECTS RET

PROJ     PROJ      MANAGER
CODE     NAME        NUM

002   GROUNDS       00006
005   BUILDING 2    00003
008   SHED          00002
018   RESEARCH      00006
037   PUB REL       00008
073   MATERIALS     00002

DTR>
```

15.2 Changing Record and File Definitions and Using New Names

This section shows you how to change record and file definitions using new names for the record and data file.

To create a new domain with two fields added to PROJECTS_REC, follow these steps:

1. Define a new domain:

```
DTR> DEFINE DOMAIN NEW_PROJECTS RET
DFN> USING NEW_PROJECTS_REC ON NWPROJ.DAT; RET
DTR>
```

2. Use the EXTRACT command to copy the old record definition to a command file:

```
DTR> EXTRACT ON TEMP PROJECTS_REC RET
DTR>
```

3. Exit DATATRIEVE and edit the command file TEMP.CMD. Remove the DELETE command from the first line of the file, change the name of the record, add the new fields, and include any other changes you want.
4. Enter DATATRIEVE again and invoke the modified command file to enter the new record definition in the data dictionary:

```
DTR> @TEMP RET
DEFINE RECORD NEW_PROJECTS_REC
 USING
01 NEW_PROJECT.
   03  PROJ_CODE        PIC 9(3) QUERY_NAME IS CODE.
   03  PROJ_NAME        PIC X(10) QUERY_NAME IS NAME.
   03  PROJ_COST        PIC 9(6)V99 EDIT_STRING IS $$$,$$9.99.
   03  MANAGER_NUM      PIC 9(5).
   03  MANAGER_NAME     PIC X(15).
;
[Record NEW_PROJECTS_REC is 41 bytes long]
DTR>
```

5. Define a data file for NEW_PROJECTS. This example creates an indexed file to replace the sequential file associated with PROJECTS:

```
DTR> DEFINE FILE FOR NEW_PROJECTS KEY=CODE RET
DTR>
```

You are now ready to transfer the data from the old domain to the new one.

15.2.1 Storing Data from All the Records in the Old Domain

This section tells you how to transfer data from all the records in the old domain. The next section explains how to transfer data from a subset of the records in the old domain.

You must first ready both domains. Ready the new domain for WRITE or EXTEND access and ready the old domain for READ access. Use the STORE statement in a FOR loop to transfer the data:

```
DTR> READY NEW_PROJECTS WRITE RET
DTR> READY PROJECTS RET
DTR> FOR PROJECTS RET
[Looking for statement]
CON> STORE NEW_PROJECTS USING RET
[Looking for assignment statement(s)]
CON> NEW_PROJECTS_REC = PROJECTS_REC RET
DTR>
```

For each field name in NEW_PROJECTS_REC that matches a field name in PROJECTS_REC, the STORE statement transfers field values from the record in PROJECTS to a record in NEW_PROJECTS. If a field in NEW_PROJECTS_REC does not match a field in PROJECTS_REC, DATATRIEVE initializes the field according to its data type and field definition (zero for numeric fields; spaces for alphabetic and alphanumeric ones).

The data file associated with NEW_PROJECTS now has records in it. When you display its contents on the terminal, you can see the values transferred from the PROJECTS domain, as well as initial values for the two new fields, PROJ_COST and MANAGER_NAME:

```
DTR> PRINT NEW_PROJECTS RET

PROJ     PROJ        PROJ     MANAGER      MANAGER
CODE     NAME        COST       NUM         NAME

002   GROUNDS          $0.00   00006
005   BUILDING 2       $0.00   00003
008   SHED             $0.00   00002
018   RESEARCH         $0.00   00006
037   PUB REL          $0.00   00008
073   MATERIALS        $0.00   00002

DTR>
```

15.2.2 Storing Data from a Subset of the Records in the Old Domain

You can create the new domain from a subset of the old domain records. You specify the limiting conditions in the RSE following the FOR statement. For example, you could have limited NEW_PROJECTS to the projects of only two managers:

```
DTR> FOR PROJECTS WITH MANAGER_NUM EQ 2, 6 RET
[Looking for statement]
CON> STORE NEW_PROJECTS USING RET
[Looking for assignment statement(s)]
CON> NEW_PROJECTS_REC = PROJECTS_REC RET
DTR> PRINT NEW_PROJECTS RET

PROJ     PROJ        PROJ     MANAGER      MANAGER
CODE     NAME        COST       NUM         NAME

002   GROUNDS          $0.00   00006
008   SHED             $0.00   00002
018   RESEARCH         $0.00   00006
073   MATERIALS        $0.00   00002

DTR>
```

15.2.3 Deleting References to the Old Domain

After you store records from the old domain in your new one, you can delete the old data file from your directory. You can continue to use your old procedures with the new record definition and data file, however, by using the REDEFINE command. As described earlier in this chapter, the REDEFINE command deletes the old domain and creates a new domain

with same name as the old domain. The domain is now defined with the new record definition and data file:

```
DTR> REDEFINE DOMAIN PROJECTS USING RET
DFN> NEW_PROJECTS_REC ON NWPROJ.DAT; RET
DTR>
```

15.3 Changing Record and File Definitions and Using Old Names

This section explains how to restructure domains when you do not want to access old data, but you want to keep the domain, record, and file names you created for the old domain. The sample statements use the domain PROJECTS.

Using an alias is the easiest way to change data structures without changing the names you use for them.

NOTE

Notice that the changes made to the PROJECTS record and file definitions are not associated with the names you want until you use the FINISH command and ready the PROJECTS domain again.

The following steps explain how to restructure domains. Steps preceded by (RSTS/E only) are necessary only if you are working on a RSTS/E system:

1. (RSTS/E only) At the system command level, rename the data file associated with the domain you are restructuring. The examples in the remaining steps assume that PROJEC.DAT has been renamed OLDPRO.DAT.
2. At the DATATRIEVE command level, use the EXTRACT command to copy the old record definition to a command file:

   ```
   DTR> EXTRACT ON TEMP PROJECTS_REC RET
   DTR>
   ```

3. At the system command level, edit the command file TEMP.CMD to add the desired field definitions. Do not remove the DELETE command from the first line of the file.
4. (RSTS/E only) At the DATATRIEVE command level, redefine the domain specifying the renamed data file:

```
DTR> REDEFINE DOMAIN PROJECTS USING RET
DFN> PROJECTS_REC ON OLDPRO.DAT; RET
DTR>
```

5. At the DATATRIEVE command level, ready the domain as an alias:

```
DTR> READY PROJECTS AS OLD_PROJECTS RET
DTR> SHOW READY RET
Ready domains:
        OLD_PROJECTS: RMS SEQUENTIAL, PROTECTED READ
DTR>
```

6. Invoke the modified command file to enter the revised record definition in the data dictionary:

```
DTR> @TEMP RET
DELETE PROJECTS_REC;
DEFINE RECORD PROJECTS_REC
 USING
01 PROJECT.
   03  PROJ_CODE        PIC 9(3) QUERY_NAME IS CODE.
   03  PROJ_NAME        PIC X(10) QUERY_NAME IS NAME.
   03  PROJ_COST        PIC 9(6)V99 EDIT_STRING IS $$$,$$9.99.
   03  MANAGER_NUM      PIC 9(5).
   03  MANAGER_NAME     PIC X(15).
;
[Record PROJECTS_REC is 41 bytes long]
DTR>
```

7. (RSTS/E only) Redefine the domain PROJECTS using your original name for the data file:

```
DTR> REDEFINE DOMAIN PROJECTS USING RET
DFN> PROJECTS_REC ON PROJEC.DAT; RET
DTR>
```

8. Define a new data file for the domain. When you ready a domain with an alias, as in Step 5, defining a new file does not interfere with the link between that readied domain and the file containing the old data. Do not use the SUPERSEDE option of the DEFINE FILE command. The following example changes the file organization from sequential to indexed:

```
DTR> DEFINE FILE FOR PROJECTS KEY = NUM RET
DTR>
```

9. Ready the domain as a different alias and specify WRITE access mode. This READY command uses the new record definition and opens the new data file created by the DEFINE FILE command:

```
DTR> READY PROJECTS AS NEW_PROJECTS WRITE RET
DTR> SHOW READY RET
Ready domains:
        NEW_PROJECTS: RMS INDEXED, PROTECTED WRITE
        OLD_PROJECTS: RMS SEQUENTIAL, PROTECTED READ
DTR>
```

10. Use the STORE statement in a FOR loop to move the data from the original domain to the new one.

    If you plan to use all the records in the domain, you simply specify the alias of the old domain in the FOR statement:

```
DTR> FOR OLD_PROJECTS RET
[Looking for statement]
CON> STORE NEW_PROJECTS USING RET
[Looking for assignment statement(s)]
CON> PROJECTS_REC = PROJECTS_REC RET
DTR>
```

    If you plan to use a subset of the records in the old domain, specify a more restrictive RSE including the alias of the old domain. The following example limits the record stream to the projects of two managers:

```
DTR> FOR OLD_PROJECTS WITH MANAGER_NUM EQ 2, 6 RET
[Looking for statement]
CON> STORE NEW_PROJECTS USING RET
[Looking for assignment statement(s)]
CON> PROJECTS_REC = PROJECTS_REC RET
DTR>
```

11. When you type FINISH and ready the domain again, you can access data as you restructured it. DATATRIEVE no longer recognizes either alias. The following example assumes that the RSE specified in the FOR loop included all the records from the domain:

```
DTR> FINISH RET
DTR> READY PROJECTS RET
DTR> SHOW READY RET
Ready domains:
        PROJECTS: RMS INDEXED, PROTECTED READ
DTR> PRINT PROJECTS RET

PROJ    PROJ        PROJ     MANAGER  MANAGER
CODE    NAME        COST       NUM      NAME

002  GROUNDS          $0.00  00006
005  BUILDING 2       $0.00  00003
008  SHED             $0.00  00002
018  RESEARCH         $0.00  00006
037  PUB REL          $0.00  00008
073  MATERIALS        $0.00  00002

DTR>
```

12. (RSTS/E only) At the system command level, delete the data file associated with your domain before you changed the file organization. Using the PROJECTS domain as an example, you would delete the file OLDPRO.DAT.

15.4 Changing the Organization of a Data File

You can also use an alias with the READY command to change only the organization of a data file associated with a domain.

The following steps make the indexed file created for PROJECTS a sequential file again:

1. (RSTS/E only) At the system command level, rename the data file associated with PROJECTS. The examples in the following steps assume that PROJEC.DAT has been renamed IDXPRJ.DAT.
2. (RSTS/E only) At the DATATRIEVE command level, redefine the PROJECTS domain specifying the renamed data file:

```
DTR> REDEFINE DOMAIN PROJECTS USING RET
DFN> PROJECTS_REC ON IDXPRJ.DAT; RET
DTR>
```

3. Ready the domain using an alias:

```
DTR> READY PROJECTS AS IDX_PROJECTS RET
DTR> SHOW READY RET
Ready domains:
        IDX_PROJECTS: RMS INDEXED, PROTECTED READ
DTR>
```

4. (RSTS/E only) Redefine PROJECTS again, using the original name of the data file:

```
DTR> REDEFINE DOMAIN PROJECTS USING RET
DFN> PROJECTS_REC ON PROJEC.DAT; RET
DTR>
```

5. Define for the domain a data file that has the organization you want. The following example defines a sequential file:

```
DTR> DEFINE FILE FOR PROJECTS RET
DTR>
```

6. Ready the domain for WRITE access using another alias:

```
DTR> READY PROJECTS AS SEQ_PROJECTS WRITE RET
DTR>
```

7. Using a STORE statement in a FOR loop, transfer records from the old domain to the new one:

```
DTR> FOR IDX_PROJECTS RET
[Looking for statement]
CON> STORE SEQ_PROJECTS USING RET
[Looking for assignment statement(s)]
CON> PROJECTS_REC = PROJECTS_REC RET
DTR>
```

8. Type FINISH. When you ready the domain again, access records in the reorganized file:

```
DTR> FINISH RET
DTR> READY PROJECTS RET
DTR> SHOW READY RET
Readied domains:
          PROJECTS: RMS SEQUENTIAL, PROTECTED READ
DTR>
```

9. (RSTS/E only) You may want to print some records to ensure that records were successfully copied to your reorganized file. Then, at the system command level, delete the data file with the organization you no longer want. Using the PROJECTS domain as an example, you would delete IDXPRJ.DAT from your directory.

Chapter 16

Editing DATATRIEVE Files

There are two basic ways to edit DATATRIEVE dictionary objects:

- At operating system command level, use your preferred editor language to create and edit command files.
- At DATATRIEVE–11 command level, use the EDIT command to invoke the EDT editor. (In DATATRIEVE–11 Version 3.3, EDT is the DATATRIEVE-resident editor. It replaces the Query Editor (QED), also known as the DATATRIEVE Editor, that was used in previous versions.)

Chapter 10 explains how to use command files. If you are accustomed to using a text editor other than EDT, you may prefer that method of changing DATATRIEVE objects.

The rest of this chapter explains:

- How to invoke EDT with the EDIT command
- How to use EDT to edit DATATRIEVE objects

This manual does not contain a detailed description of the EDT editing language. Refer to EDT documentation for full tutorial and reference information. EDT is a popular and flexible general-purpose text editor, available with all operating systems that support DATATRIEVE–11.

16.1 When to Invoke EDT with EDIT

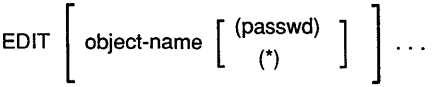
You can use the EDIT command only to modify dictionary objects that are defined in your current data dictionary. To edit DATATRIEVE commands and statements, you must store them as a procedure.

Edit with caution, especially when changing record definitions. The editor does not check for DATATRIEVE syntax errors as you edit dictionary objects. Errors show up only when you try to load in or invoke the edited dictionary object; that is, after you have exited the editor and purged the old version of the object. You might prefer to do large editing jobs in stages and thus make it easier to localize possible errors.

Changing objects with the editor does not affect domains that you currently have readied in your workspace. For example, if first you ready the domain PERSONNEL and then edit its record definition, you are not changing the record definition currently being used by the PERSONNEL domain. For the new record definition to take effect, you must finish the PERSONNEL domain and ready it again. Similarly, edits do not change a dictionary table loaded into your DATATRIEVE workspace until you release the table and refer to it again.

16.2 Invoking the Editor

You invoke the editor at the DATATRIEVE command level with the following command:



Arguments

object-name	Is the name of a dictionary object in the current data dictionary.
(passwd) (*)	Is an asterisk enclosed in parentheses (*) or the password necessary to gain C (control) access to the dictionary object. If you specify a password, you must enclose it in parentheses. If you specify (*), DATATRIEVE prompts you for the password but does not print the response on your terminal. If you omit this argument, DATATRIEVE uses your login UIC/PPN to verify that you have C (control) access privilege to the dictionary object you want to edit.

The following example invokes EDT to edit the record YACHT in the default directory. In this case, EDT displays the first line of the file and then prompts for command input:

```
DTR> EDIT YACHT RET
    1       DELETE YACHT;
*QUIT RET
DTR>
```

The EDT command QUIT terminates the EDT process without changing anything.

The EDIT command spawns a process to run EDT. If you specify one or more dictionary objects, the objects are extracted from the dictionary and copied to a .TMP file. The .TMP file is automatically displayed on the terminal, where you can edit it using the EDT language.

When you EXIT from EDT, DATATRIEVE invokes the edited .TMP file as an ordinary command file. This causes your changes to be made in the dictionary. The previous version of each edited dictionary object is purged at this time; therefore, any errors you have made during the edit session can be corrected only by updating the dictionary with another edit session. You can prevent your edits from updating the dictionary by terminating EDT with QUIT.

If you EXIT from EDT, the contents of the .TMP file will be invoked automatically, allowing you to check whether your edited object is valid.

Any editing process is subject to human error, however. If, after exiting EDT, you see error messages indicating that you have introduced syntax errors, type the EDIT command without specifying any dictionary object. DATATRIEVE–11 will spawn EDT and initiate editing on whatever .TMP file was last edited from that terminal. This allows you to recover and edit an object that was left invalid by a previous editing session at a particular terminal. .TMP files are not deleted automatically; they are retained in your directory until you delete them.

If you specify a dictionary object that does not exist, you will receive an error message. For example:

```
EDIT FOO

'FOO' has not been defined in the dictionary
```

You may sometimes see an error such as the following when you try to invoke EDIT:

```
Cannot spawn EDIT job - procedure aborting
```

You should bring such an error to the attention of the system manager.

16.3 Using EDT

The remainder of this chapter introduces the EDT language. See EDT documentation for detailed information on the EDT editing language.

EDT has a set of line mode commands (similar to the DATATRIEVE Editor used in previous versions) that you enter in response to prompts. One line mode command, CHANGE, switches you to character change mode, which allows you to edit at the character level, guided by a cursor. You can tailor EDT with specialized functions that you define in the EDT initialization file, EDTINI.EDT.

EDT is a general-purpose editor that often provides several ways to achieve a particular result. You may find that you can do all your work with just a small portion of EDT's full capability, or you may prefer to tailor the EDTINI.EDT file with many special functions.

When you invoke EDT, it can begin in either line command mode or keypad mode, depending on the contents of the EDT initialization file.

16.3.1 Line Command Mode

In line command mode, you enter all input in response to a prompt, and EDT interprets all your input as commands. The default prompt is an asterisk (*), but you can change it to any character string that you prefer.

You can move to various parts of the file by specifying line numbers or character strings. You display and alter the text of the dictionary object, and you can insert the entire contents of another file into the object you are working on. One command, CHANGE, switches you to character change mode (see Section 16.3.2).

In line command mode, EDT uses a line pointer to keep track of the current line. The line pointer points to the entire current line, not to any part of the line.

You can display the current line by typing a period (.) and pressing RETURN in response to the command prompt.

The line pointer can also point to the end of the text buffer, where you can add text to the end of the dictionary object. The symbol [EOB] marks the end of the text buffer.

Range Specification

The EDT line commands for deleting, inserting, replacing, and typing lines, and for substituting strings all contain an optional argument that specifies the range of lines on which the command operates. The range may be a single line, a series of consecutive lines, or a set of lines that contain a particular character string.

16.3.2 Character Change Mode

There are two forms of character change mode: keypad mode and character command mode. Keypad mode is the commonly-used form; character command mode must be used on terminals that do not support keypad editing. On terminals with a keypad, you can use whichever form of character change mode that you prefer.

In keypad mode, all keyboard input is treated as text, not commands. You can enter text directly into the dictionary object at the location indicated by a moving cursor. Instead of commands, you use various keypad functions to move and delete text and to change the cursor position. You leave keypad mode (and return to line command mode) by pressing CTRL/Z.

In character command mode, you enter new text as keyboard input, as in keypad mode. However, a set of special keyboard commands are used instead of keypad functions. Thus you enter both commands and new text through the keyboard. Although character command mode may seem indirect and arcane compared to keypad mode, it is worth learning. With practice, you may find it to be faster on certain text operations.

Chapter 17

Optimizing Workspace and Response Time

This chapter explains the concept of DATATRIEVE workspace. It shows you how to use memory space efficiently during a DATATRIEVE session and suggests ways to optimize DATATRIEVE performance.

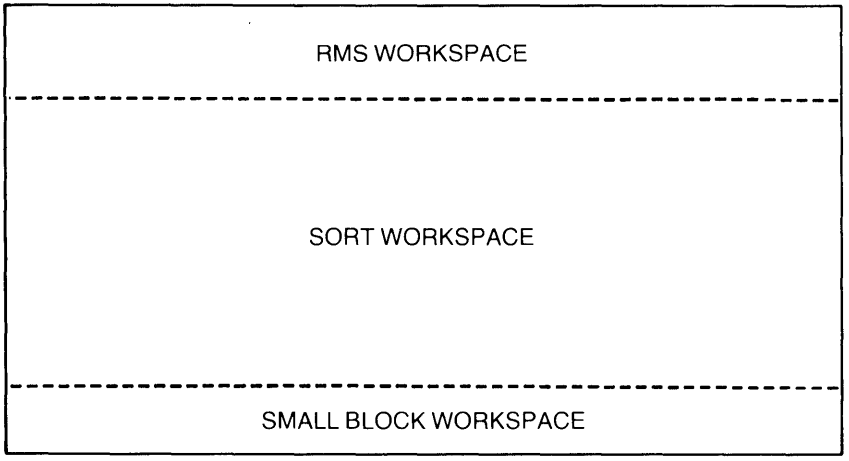
17.1 Using Workspace

Your DATATRIEVE **workspace** is the area in physical memory that is available to you during your DATATRIEVE session. Sometimes referred to as "pool space," the workspace is not the same as disk space. Workspace refers instead to the size of the current DATATRIEVE task you are performing. The maximum workspace allowed for each DATATRIEVE session is 32K words. If the task you perform requires DATATRIEVE to use more than 32K words of memory, you receive an error message, such as "Compiler storage pool exhausted," and are unable to complete your task.

17.2 Effect of READY and FINISH on Workspace

Before you ready any domains, the workspace looks like that in Figure 17–1. All the workspace is available, some of it for carrying out tasks related to Record Management Services (RMS), some for carrying out tasks related to DATATRIEVE domains, and some for sorting the records as you issue DATATRIEVE commands and statements.

Figure 17–1: Empty DATATRIEVE Workspace



The dashed lines indicate temporary boundaries. DATATRIEVE takes space from the sort workspace and allocates it to the RMS workspace and the small block workspace when they need more space. A readied domain uses space in both the RMS workspace and the small block workspace. Collections, variables, and tables all use space in the small block workspace.

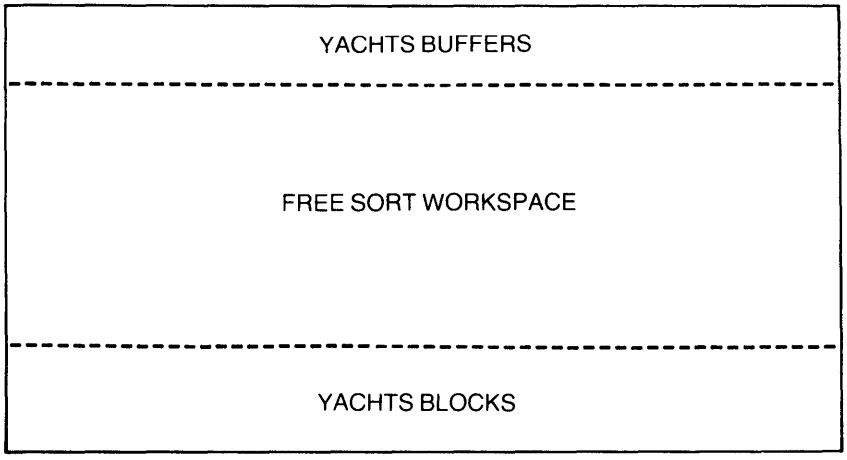
If you use the SHOW SPACE command, you see that the proportions of space used are comparable to those in the diagram. The numbers you see for your system will not be identical with those in the following example:

```
DTR> SHOW SPACE RET
                        ***Current Memory Usage***
                        Allocated       Used      Free     # of Fragments
RMS pool                 4384           4148       236       3
Small block pool          312            312         0       0
Sort pool               14024              0     14024       0
Total                   18720           4460     14260       3
DTR>
```

When using the SHOW SPACE command, pay particular attention to the total used space and the total free space. As you take up more workspace, the used space increases and free space decreases.

When you ready a domain, you begin to use up the available workspace, as shown in Figure 17–2.

Figure 17–2: Workspace with One Readied Domain



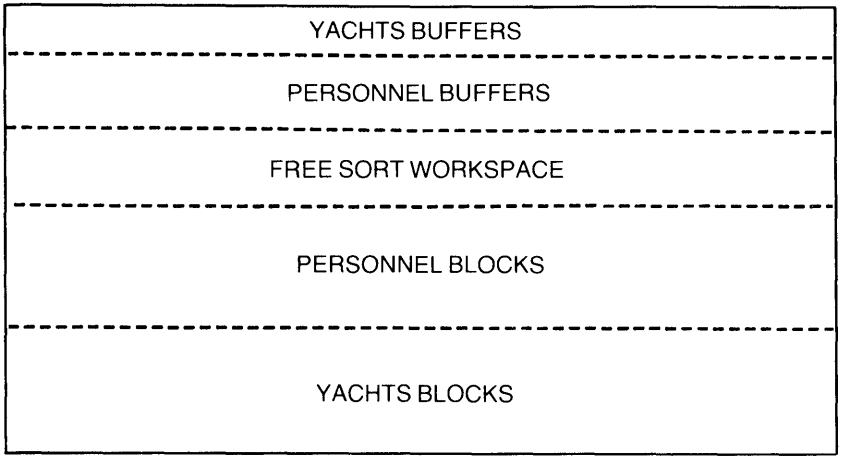
ZK-0975A-HC

If you type SHOW SPACE, you can see how the values change for current use of memory:

```
DTR> READY YACHTS RET
DTR> SHOW SPACE RET
                          ***Current Memory Usage***
                          Allocated        Used     Free     #  of Fragments
RMS pool                     4384          4292      220       1
Small block pool             2300          1588      712      15
Sort pool                   12036             0    12036       0
Total                       18720          5880    12840      16
DTR>
```

If you ready a second domain, you use additional space as shown in Figure 17–3.

Figure 17–3: Workspace with Two Readied Domains



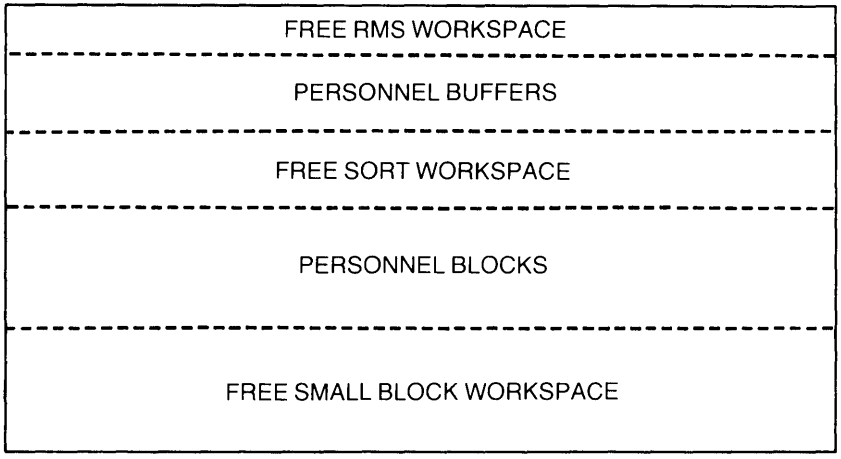
ZK-0976A-HC

Type SHOW SPACE to see values for current memory use after readying two domains:

```
DTR> READY PERSONNEL RET
DTR> SHOW SPACE RET
                        ***Current Memory Usage***
                        Allocated              Used    Free     # of Fragments
RMS pool                   4944                4388     220       5
Small block pool           3416                2656     760      21
Sort pool                 10360                   0   10360       0
Total                     18720                7044   11676      26
DTR>
```

If you finish a domain, you free the workspace that the domain was occupying. The sort workspace recovers free space only if the free space is adjacen to the sort workspace. If you free space in the interior of the RMS workspac or small block workspace, then it remains in that workspace as a fragment, as shown in Figure 17–4.

Figure 17–4: Workspace When You Finish First Readied Domain



ZK-0977A-HC

The following SHOW SPACE command illustrates that releasing the first readied domain does not increase the amount of free sort workspace:

```
DTR> FINISH YACHTS RET
DTR> SHOW SPACE RET
                          ***Current Memory Usage***
                          Allocated        Used    Free    # of Fragments
RMS pool                     4944          4244     700       5
Small block pool             3416          1380    2036      12
Sort pool                   10360             0   10360       0
Total                       18720          5624   13096      17
DTR>
```

You use the sort workspace each time you specify the order of a record stream with the SORTED BY clause in an RSE or a collection with the SORT statement. DATATRIEVE immediately seizes space from the sort workspace and releases it when it finishes the sort. DATATRIEVE sorts slowly when it has little space to work with, so keeping the sort workspace as large as possible saves computing time. If the sort workspace is too small, DATATRIEVE returns the message:

```
Sort workspace exhausted
Execution failed
```

17.3 Techniques to Optimize Workspace

The following techniques for readying and finishing domains can help you optimize your workspace:

- Try to reduce the number of files you have open at a time. If you no longer need a domain, FINISH it.
- Pay attention to the order in which you open the files. As the illustrations show, when you finish a domain you do not always free a complete, contiguous block of workspace. If you finish the first domain before you finish the second, your workspace looks like that in Figure 17–4. It is best, therefore, first to ready the domains you plan to use the longest and then ready and finish those you need for only a short time.

You can also save space in the way you define your records too. Each time you define a field, you add overhead cost in addition to the size of the item being stored. Using EDIT_STRING, QUERY_HEADER, and QUERY_NAME clauses also adds to the storage space your record requires. Use the following guidelines when defining your record:

- Keep record definition clauses short, in particular EDIT_STRING, QUERY_HEADER, and QUERY_NAME clauses. Do not use any of these clauses unnecessarily.
- Use short names and eliminate unnecessary fields. Unnecessary group fields are a particular waste of space.
- When you use FILLER, try to combine two or more elementary fields into one FILLER field.

Additional space-saving techniques are as follows:

- Release collections, tables, and variables that you do not need.
- Avoid using long BEGIN-END blocks, and the THEN connector to form compound statements. DATATRIEVE compiles the complete statement all at once, at the cost of workspace.
- Use a small file bucket size. The file bucket size determines the size of the RMS buffers needed. A bucket size of 8 would use approximately half of DATATRIEVE workspace and would cause you to run out of space repeatedly. A bucket size of 1 or 2 reduces the requirements for DATATRIEVE workspace.

- For the REPORT statement, first use the FIND statement to form a collection and the SORT statement to sort and then report on the sorted collection. Avoid including the SORTED BY clause in the REPORT statement. The REPORT and SORT statements both require large amounts of workspace.

17.4 Techniques to Optimize Response Time

As mentioned earlier, DATATRIEVE performs best when it has adequate space to work with. Therefore, when you keep the sort pool in your workspace as large as possible, you improve response time. The following sections suggest other ways you can improve DATATRIEVE performance.

17.4.1 Using the ALLOCATION Option of the DEFINE FILE Command

If you know approximately how many blocks of storage your data file will require and you want to enter records quickly, you can use the ALLOCATION option in the DEFINE FILE command to reserve contiguous storage space for the file. This option is particularly useful if you know your data file will be large. The format for the ALLOCATION option is ALLOCATION = n, where n is the number of blocks you want as the initial allocation for the file.

17.4.2 Using Keyed Access Efficiently

DATATRIEVE allows you to define indexed or sequential files for your data. Sequential files require less storage, but DATATRIEVE must search records one by one according to their physical order in the file. This organization may be optimal in certain cases. For example, a domain's records may contain a field for the current date, so you may want records physically arranged in the order in which you stored them. For instance, you are likely to want to have sequential access to banking transactions. If you access groups of records in chronological order, you might find sequential organization efficient.

In other cases, your access needs may not be suited to sequential organization. You may need to access a group of records that are distributed throughout the file. If you have stored the records in a sequential file, DATATRIEVE may have to read all the records to find the one or two that you request. In this case, indexed file organization is probably a better choice. Although indexed files require more storage, DATATRIEVE can

search indexed files quickly to find the records you want, if you base the search on a key field.

When defining data, try to decide which field of the record you are likely to name most often in queries. Make that field the primary key if its value is likely to be unique for each record. For example, if you are setting up a personnel domain, you might predict that most users seek information based on employee ID. In that case, make the ID field the primary key.

By default, primary key values are unique. That is, the primary key value by itself is enough to identify a record. It is legal in DATATRIEVE to specify that primary keys can have duplicate values. However, allowing duplicate primary key values is not recommended; having too many duplicate key values slows DATATRIEVE searches that are based on key fields.

If the leading candidate for the primary key does not uniquely identify the record, find another field such that the two fields combined can uniquely identify the record. You can then designate a group field, encompassing the two fields, as the primary key. For example, in the YACHTS domain, the group field TYPE (consisting of BUILDER and MODEL) is the primary key, uniquely determining records in YACHTS.

Avoid using a field defined as USAGE IS DATE as a key field. The value stored in a date field is larger than the maximum value allowed for a key by RMS. Therefore, specifying a date field as a key provides no response time advantage when you base a query on a comparison of the field value and the range of values (for example, LT, GT, LE, GE, and BETWEEN). If you base your query of an indexed data field on an equality comparison, DATATRIEVE uses the index and performs faster than if it did a sequential search of the records.

After organizing your indexed file and storing records in the file, you should structure DATATRIEVE queries to take advantage of keyed access. A query is a request for DATATRIEVE to identify all the records that satisfy a specified condition. Not all the DATATRIEVE queries you can formulate use keyed access to indexed files. The following sections tell you how to produce queries that give you the fastest response time.

17.4.2.1 Using EQUAL Rather than CONTAINING

A Boolean expression that tests records with the EQUAL (=) relational operator is more efficient than a Boolean with CONTAINING (CONT) when the expression refers to a key field. For example, compare the following two queries:

```
DTR> PRINT YACHTS WITH BUILDER = "PEARSON" RET
DTR> PRINT YACHTS WITH BUILDER CONT "PEARSON" RET
```

Although both queries yield the same results, the first query is twice as fast as the second one.

To resolve the first query, DATATRIEVE conducts a fast search through the index to retrieve the desired records. In the second case, DATATRIEVE must search through the values of BUILDER looking for matches with the string following CONT. DATATRIEVE must check all substrings of each BUILDER value that are equal in length to the string specified in the Boolean.

To take advantage of the increased efficiency of EQUAL (=), you must specify a value that matches the field value exactly. EQUAL (=) is case sensitive but CONT is not. In the last example, if a record had the value "Pearson" for BUILDER, only the second query would find the record.

To get around the problem of case sensitivity, you might consider using only uppercase letters when entering data. Otherwise, to use the EQUAL operator, you must remember the case of each character of a field value.

17.4.2.2 Choosing Domains or Collections as Record Sources

When you form a collection, DATATRIEVE can no longer use key-based access for retrieving records. In most cases, you get the best performance on key-based queries when you specify a domain rather than a collection in the RSE.

Furthermore, if you form a query that relates to more than one record source, all but the last source specified is an implied collection. Therefore, you cannot have key-based access for any but the last record source in a relational query. When you use nested FOR loops and specify domains and key fields in each loop, for example, DATATRIEVE can use a key-based index only for evaluating the RSE in the last FOR loop.

If all other conditions are equal, it is better to use a domain name than a collection name in the last position of a key-based relational query. But there is one more factor to consider: collections are efficient to use if you need to refer back to the same group of records in the same DATATRIEVE session. This is especially true if the collection is much smaller than the data file from which it is formed. In such a case, you may get better performance by forming and naming a collection so that DATATRIEVE does not have to retrieve the same group of records over and over again.

To summarize, you gain efficiency with a domain when you can use keyed access. You gain efficiency with a collection if you reduce the number of times DATATRIEVE must isolate the same small group of records from a large body of records.

17.4.2.3 Ordering the Domains in Nested FOR Loops

If one FOR loop must process many more records than the others and all have the same key field, include the larger record stream in the last (inner) FOR loop. Similarly, if only one FOR loop can use keyed access, make sure that FOR loop is the last, or innermost, loop you specify.

17.4.2.4 Restoring Indexed Files That Are Often Modified

If you add, erase, or change many records in an indexed file, that file can degrade DATATRIEVE performance. When you have erased many records from an indexed file, records in the file can occupy many noncontiguous areas of the disk. This slows down record access. When you add records to a file or change many values in key fields, the index for the file can be split over many noncontiguous areas of the disk. This also slows down record access. The more keys you define for each file, the more quickly DATATRIEVE response time degrades when you make many changes to the file.

If you have a DATATRIEVE performance problem and you have made many modifications to your indexed file, restructuring your domain might improve DATATRIEVE response time. When you restructure a domain, you are recreating the data file. When the new file is created, the records and index are stored as much as possible on contiguous areas of the disk.

Follow the same steps given in Chapter 15 for reorganizing a data file. In this case, however, specify the same organization for your new file rather than a different one. If you have defined many different keys for your file, you might want to eliminate any keys that you seldom use in queries. If you define as keys only those fields that you often use in queries, modifying your file has less effect on DATATRIEVE response time.

You can also restructure files and indexes using the RMS–11 utilities CONVERT and IFL. These utilities are explained in the RMS–11 manuals in your operating system documentation set. The DATATRIEVE–11 utility program QCPRS, explained in Chapter 20 of this manual, can be used to restructure files and indexes.

17.4.3 Avoiding Nested FOR Loops Followed by a Conditional Statement

Try to avoid using nested FOR loops to control the execution of a conditional statement. The following example is extremely inefficient:

```
DTR> FOR A IN OWNERS RET
CON> FOR YACHTS RET
CON> IF TYPE = A.TYPE RET
CON> THEN PRINT BOAT, A.NAME RET
```

Wherever possible, include conditional tests as Boolean expressions within one of the RSEs. This effectively limits the number of records that DATATRIEVE has to process. The following example works much more efficiently than the preceding one:

```
DTR> FOR A IN OWNERS RET
CON> FOR YACHTS WITH TYPE = A.TYPE RET
CON> PRINT BOAT, A.NAME RET
```

Chapter 18

Controlling Output

When you invoke DATATRIEVE, several characteristics are set that control your display of input and output:

- The number of columns in an output display (COLUMNS_PAGE)
- The way DATATRIEVE responds to an ABORT statement ([NO] ABORT)
- The presence or absence of "Looking for . . . " prompts ([NO] PROMPT)

You can change these characteristics at any time during a DATATRIEVE session by using the forms of the SET command discussed in the following sections.

18.1 Changing the Columns-Page Setting

The default for the columns-page setting is 80 characters, the width of most video display screens. You can change this setting to fit your application and terminal characteristics.

18.1.1 Increasing the Columns-Page Setting

You may want to increase the columns-page setting on your video terminal or hardcopy terminal if you want to display detail lines more than 80 characters long. If you have a Digital-supported video terminal and your command language is DCL, do the following:

1. Use the DCL SET TERMINAL command to tell your system to increase the width of lines it can send you:

```
$     TERMINAL/WIDTH=132 RET
```

2. Use the SET COLUMNS_PAGE command to increase the length of the line DATATRIEVE can display on your terminal. The maximum limit on the columns-page setting is 255.

```
DTR> SET COLUMNS_PAGE = 132 RET
```

Whatever the column setting on your terminal, you can continue a long input line by using a hyphen (-) continuation character at the end of the line. When you use a hyphen, DATATRIEVE does not check the syntax of your input until you press RETURN after a line that does not end in a hyphen. If the line you want to extend ends with a complete word, separate the hyphen from the word by entering a space. Otherwise, DATATRIEVE considers the characters at the beginning of the next line to be part of the same character string.

18.1.2 Decreasing the Columns-Page Setting

To decrease the number of columns displayed, enter a SET COLUMNS_PAGE command:

```
DTR> SET COLUMNS_PAGE = 60 RET
DTR>
```

Decreasing the columns-page setting may cause problems when you display your output, however. If one of the elements in a print line is longer than the columns-page setting, DATATRIEVE displays an error message:

```
DTR> SET COLUMNS_PAGE = 15 RET
DTR> PRINT "1234567890123456" RET
Print object too large for line width

DTR>
```

Notice that the columns-page setting does not affect the length of your input lines or the messages you receive from DATATRIEVE.

Reducing the columns-page setting can also distort the display of a record as in the following example:

```
DTR> READY YACHTS RET
DTR> PRINT FIRST 1 YACHTS RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL      RIG   ALL   WEIGHT BEAM  PRICE

 ALBERG       37 MK II   KETCH   37    20,000  12  $36,951

DTR> SET COLUMNS-PAGE = 12 RET
DTR> PRINT FIRST 1 YACHTS RET
```

```
MANUFACTURER

 ALBERG
37 MK II
KETCH
 37
20,000  12
$36,951

DTR>
```

18.1.3 Determining the Number of Columns You Need for a Print Line

Several considerations determine the number of spaces, or columns, needed for each print object in the output line:

- The actual length of the character string literal or numeric literal, or the length of the field or variable as specified by the record definition or the DECLARE command
- The length of the field name, query name, query header, or longest segment of the query header
- The length of the edit string

The longest of these determines how many columns you need.

To prevent crowding, DATATRIEVE also adds one space to the length of all fields except the last one on a line. To print a record from the YACHTS domain requires a minimum of 57 columns:

- MANUFACTURER takes 13 columns; the query header is 12 characters long.
- MODEL takes 11 columns; the field is defined as 10 characters long: PIC X(10).
- RIG takes 7 columns; the field is defined as 6 characters long: PIC X(6).
- LENGTH_OVER_ALL takes 7 columns; the longest segment of the query header (LENGTH) is 6 characters long.
- DISPLACEMENT takes 7 columns; both the query header (WEIGHT) and the edit string (ZZ,ZZ9) are 6 characters long.
- BEAM takes 5 columns; the query header (BEAM) is 4 characters long.
- PRICE takes 7 columns; the edit string ($$$,$$$) is 7 characters long, but no column is added because PRICE is the last field in the detail line.

If you set the columns to 56, the PRICE field no longer fits on one line. DATATRIEVE displays the price on the following line and omits the PRICE header:

```
DTR> SET COLUMNS_PAGE = 56 RET
DTR> PRINT FIRST 1 YACHTS RET

                                  LENGTH
                                   OVER
MANUFACTURER    MODEL      RIG     ALL    WEIGHT BEAM

 ALBERG       37 MK II    KETCH   37     20,000  12
$36,951

DTR>
```

18.2 Using the SET ABORT Statement

When DATATRIEVE executes an ABORT statement in a command file or procedure while SET NO ABORT is in effect, it affects only the compound statement containing the ABORT statement. If SET ABORT is in effect, DATATRIEVE terminates the remainder of the command file or procedure. The same rules apply if you enter CTRL/Z in response to a prompt.

If DATATRIEVE encounters a syntax or logical error in a command file or procedure, it returns you to the DTR> prompt whether or not you have used SET ABORT. SET NO ABORT is the default setting when you invoke DATATRIEVE.

See Chapters 9 and 10 for a discussion of using ABORT and NO ABORT in controlling procedures and command files.

18.3 Using the SET PROMPT Statement

When you invoke DATATRIEVE, SET PROMPT is in effect. If you press RETURN before finishing a command or statement, DATATRIEVE prompts you for the remaining required elements of that command or statement.

The following sequence of commands and statements shows how DATATRIEVE responds when SET PROMPT is in effect. After the line of text indicates the next required element, DATATRIEVE displays the CON> (continuation) prompt. As long as the syntax of a command or statement is incomplete, DATATRIEVE uses CON> to tell you it is ready for further input.

```
DTR> READY RET
[Looking for Dictionary Element]
CON> YACHTS RET
DTR> FIND RET
[Looking for "FIRST", domain name, or collection name]
CON> FIRST RET
[Looking for a value expression]
CON> 1 RET
[Looking for collection or domain name]
CON> YACHTS RET
[1 Record found]
DTR>
```

Notice that DATATRIEVE stops prompting as soon as you enter elements that comprise a syntactically complete command or statement. For example, READY YACHTS is complete, and DATATRIEVE does not prompt for any further element. Similarly, when you enter FIND FIRST 1 YACHTS, DATATRIEVE does not prompt you for a Boolean expression or a SORTED BY clause.

When SET NO PROMPT is in effect, DATATRIEVE does not display the text about the next required element. It does, however, use the CON> prompt when the syntax is incomplete. The following example is identical to the previous one but has SET NO PROMPT in effect:

```
DTR> SET NO PROMPT RET
DTR> READY RET
CON> YACHTS RET
DTR> FIND RET
CON> FIRST RET
CON> 1 RET
CON> YACHTS RET
[1 Record found]
DTR>
```

Note that SET NO PROMPT does not suppress the messages DATATRIEVE displays about the results of commands and statements.

Chapter 19

Controlling Access to Dictionary Objects

To supplement your operating system protection, DATATRIEVE uses access control lists (ACLs) to protect your data and dictionary definitions. An ACL, stored in the data dictionary, regulates user access to an object in the data dictionary.

Every dictionary object (domain, record, procedure, and table) has an associated ACL. This chapter describes the contents of an ACL and the commands you use to maintain the ACL. It also shows a strategy to help ensure the integrity of your data.

Carefully maintained ACLs can be effective against unauthorized browsing through files and accidental corruption of the dictionary or data. Use them to improve the overall security system for your installation.

19.1 Contents of an Access Control List

An ACL consists of one or more entries. Each entry contains the following information:

- A sequence number
- A lock type
- A key
- One or more access privileges

Figure 19–1 shows a sample ACL containing three entries that illustrate the parts and options in an ACL. The table within Figure 19–1 identifies the parts of each entry.

Figure 19–1: Sample Access Control List

```
DTR> SHOWP SAMPLE RET
        1,UIC, [254,203], "RWEMC"
        2,PW, "SWORDFISH", "RM"
        3,UIC, [*,*], "RE"

Sequence  Lock  Key  Privileges
Number

1               UIC    [254,203]   RWEMC

2               PW     SWORDFISH   RM

3               UIC    [*,*]     RE
```

19.1.1 Sequence Numbers

Sequence numbers are sequential integers beginning with 1 that DATATRIEVE assigns to identify ACL entries. Use sequence numbers to identify entries you are adding to or deleting from the ACL.

19.1.2 Lock Types

Each entry in the ACL has a lock type that indicates whether you access the associated dictionary object by specifying a password or by using a UIC/PPN. There are two lock types: PW for password, and UIC for UIC/PPN. (A UIC/PPN represents an operating system account number.)

19.1.3 Keys

DATATRIEVE uses keys to identify you when you request access to dictionary objects. When you attempt to access a dictionary object, you must provide the correct password for a PW lock or own the correct UIC/PPN for a UIC lock.

19.1.3.1 Password Keys

If the lock type is PW, the key is a 1- to 10-character password. You can use any character from the ASCII character set except the dollar sign ($). Examples of legal passwords include: FISH-FRY, SWORDFISH, 1234, and PASSWD–9.

To execute a command or statement that requires access privileges protected by a password, you include the password directly in the command or statement. For example, the domain YACHTS might contain only one ACL entry:

```
1                PW   SWORDFISH          RWEMC
```

To delete the YACHTS domain definition from the dictionary, you can specify the password enclosed in parentheses in the DELETE command:

```
DTR> DELETE YACHTS (SWORDFISH); RET
```

For security reasons, you can also suppress the display of the password on your terminal. To do this, type an asterisk enclosed in parentheses (*) instead of the password. DATATRIEVE then prompts for the password but does not display the password when you enter it:

```
DTR> DELETE YACHTS (*); RET
Enter password for YACHTS:          RET
```

This technique for specifying a password is especially useful if you have a hardcopy terminal.

19.1.3.2 UIC Keys

If the lock type is UIC (for UIC/PPN), the key is an account number known to the operating system. Under RSTS/E systems, an account number is called a project-programmer number, or PPN. Under other operating systems, the account number is called the user identification code, or UIC. The UIC/PPN consists of a three digit octal group number followed by a three digit octal user number. The numbers are separated by a comma and enclosed in brackets, for example [253,201].

A UIC/PPN lock can include specific numbers, asterisks, or a combination of an asterisk and a number. Asterisks allow all users or only users in a specified group to access a dictionary object. The following are valid ACL lock entries:

- [253,201], allowing only the user with the account number [253,201] to have access
- [253,*], allowing any user with group number 253 to have access
- [*,*], allowing any user to have access

For the UIC lock type, you do not include a UIC or PPN in a command or statement. Rather, DATATRIEVE verifies that you have access by checking the UIC/PPN you used to log in. For example, if the ACL for the procedure BIG-YACHTS contains just the following entry, then you must log in under [253,201] in order to access the procedure:

```
1  UIC  [253,201]   RWEMC
```

If the ACL contains the following entry, you can access the procedure regardless of your UIC/PPN:

```
1  UIC   [*,*]  RWEMC
```

Because you cannot use a password when invoking a procedure, you may want to use the UIC/PPN lock type to protect procedures.

NOTE

On RSX and VAX–11 RSX systems, a UIC must be in the range of [0,0] to [377,377]. On RSTS systems, a UIC must be in the range of [0,0] to [256,256].

19.1.4 Access Privileges

An access privilege determines the access you have to the associated dictionary object. You designate an access privilege by a single letter, a string of letters, or a space. Table 19–1 contains a list of access privileges and their description.

Table 19–1: Access Privileges

Access Privilege	Description
R	READ. The user can SHOW or EXTRACT the associated dictionary object. For a domain, the user can ready the domain for READ access only.
W	WRITE. The user can ready the domain for READ, EXTEND, MODIFY, or WRITE access to retrieve, modify, store, or erase records.
E	EXTEND or EXECUTE. For a domain, the user can ready the domain for EXTEND access only to store records. For a procedure, the user can execute the procedure. For a table, the user can refer to the table (using VIA or IN clauses). The user must have E access to a record to ready the associated domain.
M	MODIFY. The user can ready the domain for READ or MODIFY access to read or change records in the domain but not to add or delete.
C	CONTROL. The user can issue the commands DEFINEP, DELETE, REDEFINE, DELETEP, EDIT, and SHOWP.
Spaces	No access. The user cannot access the dictionary object.

Each entry in an ACL can include from one to five access privileges or designate no access. A space indicates that no access is permitted to a user with the corresponding key. For example, the single letter R specifies that a user with the corresponding key has read access only. The letters RW specify that the user has both read and write access. Full access, designated by RWEMC, allows a user complete access to the dictionary object. If you specify a space, the user has no access to the dictionary object.

Each access privilege allows you to issue certain commands and statements. For example, with R privilege, you can issue an EXTRACT command or ready a domain for read access. If you ready a domain to READ, you can then use commands and statements that display and manipulate the domain database. With only R privilege, you cannot ready a domain to WRITE, MODIFY, or EXTEND. Therefore, you cannot add to, delete, or modify data.

Table 19–2 contains a list of commands that you can issue if you are granted the corresponding access privilege. The table also shows the statements you can use after issuing the command.

Table 19–2: Commands/Statements by Privilege

Privilege for Domain	Commands Permitted	Query Statements Permitted
R	EXTRACT	FIND PRINT
	READY . . . READ	SELECT SORT SUM
E	READY . . . EXTEND	STORE
M	READY . . . READ	FIND PRINT SELECT SORT SUM
	READY . . . EXTEND	STORE
	READY . . . MODIFY	FIND MODIFY PRINT SELECT SORT SUM
W	READY . . . READ	FIND PRINT SELECT SORT SUM
	READY . . . EXTEND	STORE
	READY . . . MODIFY	FIND MODIFY PRINT SELECT SORT SUM

(continued on next page

Table 19–2 (Cont.): Commands/Statements by Privilege

Privilege for Domain	Commands Permitted	Query Statements Permitted
	READY . . . WRITE	ERASE FIND MODIFY PRINT SELECT SORT STORE SUM
C	DEFINEP	–
	DELETE	–
	REDEFINE	–
	DELETEP	–
	EDIT	–
	SHOWP	–

NOTE

You must have E access to the associated record definition to ready a domain.

The meaning of E privilege differs depending on the dictionary object to which an ACL corresponds. In an ACL for a domain, E means extend privilege; that is, the user can store records in a file or extend it. In an ACL for a table, procedure, or record definition, E means execute privilege. The user can use a table or procedure, or ready the domain associated with the record definition.

Only users with C (control) access to a dictionary object can directly access its associated ACL or edit or delete the dictionary object. However, any user with a group (or project) code of 1 (that is, a UIC/PPN in the form [1,n]) is automatically granted C (control) access to all ACLs.

19.2 Creating Access Control Lists

When you define a dictionary object, DATATRIEVE automatically creates ai ACL for that object. The ACL initially contains only one entry, a UIC/PPN that is granted full access privileges to the dictionary object. The specific UIC/PPN stored in the ACL is installation-dependent and is determined when the DATATRIEVE software is installed. The entry can be in one of th following formats:

[m,n]
Full access privileges are granted to any user with the same UIC/PPN as the creator of the dictionary definition. For example, if you log in under [253,201] and create a procedure definition, then an entry for [253,201] is stored in the access control list for the procedure. Only users with a UIC/PPN of [253,201] can access the procedure.

[m,*]
Full access privileges are granted to any user with the same group (or project) code (m) as the creator of the definition. For example, if you log in under [253,201] and create a procedure definition, then an entry for [253,*] is stored in the access control list for the procedure. Any user with a group (or project) code of 253 (such as [253,222]) also has full access privileges to the procedure.

[*,*]
All DATATRIEVE users, regardless of their UIC/PPN, are granted full access to the dictionary object.

Regardless of the default UIC/PPN stored in the ACL, the creator of the definition always has full access privileges (RWEMC) to the dictionary objec at the time he or she creates the definition.

If you have C access to a dictionary object, you can grant privileges to additional users or further restrict the use of the dictionary object by changing its ACL. The commands you use to change the table are summarized later i this chapter and described fully in the *DATATRIEVE–11 Reference Manual.*

19.3 Processing Access Control Lists in DATATRIEVE

DATATRIEVE checks the appropriate access control list to verify that you have access privilege whenever you use a table, invoke a procedure, or issue one of the following commands:

- DEFINEP
- DELETE
- DELETEP
- EDIT
- EXTRACT
- READY
- REDEFINE
- SHOW
- SHOWP

To verify that you have the correct access privilege, DATATRIEVE searches the ACL, checking each entry until it finds a match between the ACL key and your UIC/PPN or between the password you supply and one in the ACL. DATATRIEVE searches the table in the following sequence:

1. DATATRIEVE stores your UIC/PPN and determines if you are a privileged user. A privileged user is one with a group (or project) code of 1 (that is, a UIC/PPN in the form [1,n]). At a minimum, DATATRIEVE grants a privileged user C (control) access. It may grant additional privileges, depending on the results of the remaining steps.
2. DATATRIEVE checks the first entry in the access control list.

   If the lock type is PW, DATATRIEVE checks to see if you specified a password in the command. If you did and the password matches the entry's key, DATATRIEVE stops searching the access control list and grants you the privilege or privileges listed in the entry. If the password does not match or you did not specify a password, DATATRIEVE performs the next step.

   If the lock type is UIC, DATATRIEVE checks to see if your UIC/PPN matches the key. If it does, DATATRIEVE stops searching the list and grants you the privileges listed in the entry. If the UICs do not match, DATATRIEVE performs the next step.
3. DATATRIEVE checks the next entry in the list, following the same procedure as in the previous step.

When there are no more entries, DATATRIEVE denies access to the dictionary object and rejects your command or statement.

The following examples show how DATATRIEVE handles some user requests. The examples use the following ACL for the domain YACHTS:

```
DTR> SHOWP YACHTS RET
        1,UIC, [253,201], " "
        2,UIC, [214,217], "CW"
        3,PW, "FISH-FRY", "M"
        4,UIC, [*.*], "R"
DTR>
```

A user with UIC/PPN [253,201] enters a SHOW command to look at the definition of the YACHTS domain. DATATRIEVE checks the user's UIC/PPN and finds it as the first entry. Because no access privilege is granted, access to the domain is denied to the user:

```
DTR> SHOW YACHTS
Access denied to dictionary resource "YACHTS"
DTR>
```

A user with UIC/PPN [214,217] readies YACHTS for write access. The first match in the ACL (at entry 2) grants the user write (and control) privilege. The READY command executes:

```
DTR> READY YACHTS WRITE RET
DTR>
```

A user with UIC/PPN [253,201] tries to ready YACHTS for modify access by including a password in the READY command. Because the entry in the ACL that contains the password FISH-FRY appears after the entry denying all privileges to the user, modify access to YACHTS is denied:

```
DTR> READY YACHTS MODIFY (FISH-FRY) RET
Access denied to dictionary resource "YACHTS"
DTR>
```

A user with UIC/PPN [234,231] issues the same command as in the previous example. Because the user does not have the UIC/PPN that is denied all access and has included the correct password key for modify access, the READY command executes:

```
DTR> READY YACHTS MODIFY (FISH-FRY) RET
DTR>
```

19.4 Maintaining an Access Control List

To maintain an ACL that implements your security strategy, you must have C (control) access to the dictionary object associated with the ACL. Control access allows you to display an ACL and add or delete ACL entries.

19.4.1 Guidelines for Ordering Entries

When you add entries to an ACL, the order of entries in the ACL controls the access to the dictionary object. Place the most restrictive entries first in an ACL and the least restrictive entries last. The most restrictive entries completely deny access to a specific UIC/PPN, while the least restrictive entries allow access by any UIC/PPN.

The following rules apply when adding entries to the ACL:

- Place entries that deny all privileges first. Use a lock type UIC (instead of PW) for these entries.
- Place restrictive entries that limit access to a specific UIC/PPN next.
- Place the less restrictive entries (such as those requiring a password) next.
- If access is allowed for any UIC/PPN (that is, if the key is [*,*]), place its entry last in the list.

19.4.2 Assigning Privileges

If you know which commands or statements you want to permit a user to issue, use Table 19–3 to find the privileges you must assign to that user.

Table 19–3: Privilege Requirements by Command/Statement

Command Statement	Privilege Required
DEFINEP	C privilege for the dictionary object
DELETE	C privilege for the dictionary object
DELETEP	C privilege for the dictionary object
EDIT	C privilege for the procedure or table

(continued on next page)

Table 19–3 (Cont.): Privilege Requirements by Command/Statement

Command Statement	Privilege Required
EDIT ADVANCED	C privilege for the domain or record
ERASE	W privilege for the domain and E privilege for the associated record
EXTRACT	R privilege for the dictionary object
READY . . . READ	R, W, or M privilege for the domain and E privilege for the associated record
. . . WRITE	W privilege for the domain and E privilege for the associated record
. . . MODIFY	M or W privilege for the domain and E privilege for the associated record
. . . EXTEND	E, W, or M privilege for the domain and E privilege for the associated record
SHOW domain-name record-name proc-name table-name	R privilege for the dictionary object
SHOWP	C privilege for the dictionary object associated with the ACL

Use care when assigning the W privilege, particularly if a more restrictive privilege (such as R or M) would suffice. The W privilege allows the user to perform the same functions as the R, M, and E privileges but also allows the user to issue the ERASE command to delete records.

19.4.3 Displaying an Access Control List

Use the SHOWP command to display an ACL:

SHOWP object-name [(passwd) | (*)]

19.4.4 Adding Entries to an Access Control List

Use the DEFINEP command to add an entry to an ACL:

DEFINEP object-name [(passwd) | (*)] sequence-no, { PW, new-passwd | UIC, [m,n] } ,priv

Before adding any entry to an ACL, display the ACL using SHOWP. The following example illustrates adding one entry to an ACL:

```
DTR> SHOWP YACHTS (FISH-FRY) RET

          1,PW, "FISH-FRY", "RWEMC"

DTR> DEFINEP YACHTS (FISH-FRY) 2, UIC, [201,213], R RET

DTR> SHOWP YACHTS (FISH-FRY) RET

          1,PW, "FISH-FRY", "RWEMC"
          2,UIC, [201,213], "R"
DTR>
```

When you add an entry to an ACL, DATATRIEVE renumbers the entries that follow it so that their numbers occur in proper sequence.

19.4.5 Deleting Entries from an Access Control List

The DELETEP command deletes one entry from an ACL:

DELETEP object-name [(passwd) | (*)] sequence-number

Use the SHOWP command before deleting an entry to verify that you are deleting the correct entry. For example:

```
DTR> SHOWP YACHTS (FISH-FRY) RET

        1,PW, "FISH-FRY", "RWEMC"
        2,UIC, [201,213], "R"

DTR> DELETEP YACHTS (FISH-FRY) 2 RET
DTR> SHOWP YACHTS (FISH-FRY) RET

        1,PW, "FISH-FRY", "RWEMC"
DTR>
```

After you delete an entry, DATATRIEVE renumbers the entries so that they are sequential, beginning with 1.

An ACL must have at least one entry. DATATRIEVE does not allow you to delete an entry when that entry is the only one in the ACL.

If you delete all entries that have C (control) privilege, there is only one way to change the ACL. Log in using a privileged UIC/PPN, invoke DATATRIEVE, and use the DEFINEP command to create an entry that gives one or more users C privilege.

Chapter 20

Maintaining Data Dictionaries

A data dictionary holds domain, record, procedure, and table definitions. The items that you define in a data dictionary are called **dictionary objects**. DATATRIEVE supplies you with a default dictionary called QUERY.DIC. When you invoke DATATRIEVE, your current dictionary is QUERY.DIC. You can keep all your data definitions in QUERY.DIC, but it is orderly and efficient to store related definitions in separate dictionaries. You can create other data dictionaries with the CREATE DICTIONARY command.

You can change from one dictionary to another, and you can display general and specific information about your current dictionary. You can also display information about readied domains, established collections, tables currently in memory, and selected records.

You can transfer definitions stored in one dictionary to another dictionary by copying the definitions you want to a command file. You then set the destination dictionary as the current dictionary and execute the command file to store the definitions.

You can also edit dictionary definitions. If you need to make changes to an existing procedure or table, use the EDIT command to invoke EDT, as described in Chapter 16. If you need to make extensive modifications to a procedure or table or want to edit a domain or record definition, you might prefer to copy the definition to a command file, exit from DATATRIEVE, edit the command file with the editor of your choice, and return to DATATRIEVE to execute the command file and store the new definition.

The following sections discuss dictionary maintenance in detail.

20.1 Displaying Dictionary Objects

You can list the names of all domains, records, procedures, and tables defined in the current data dictionary with the SHOW ALL command. SHOW ALL also lists the names of any established collections and readied domains:

```
DTR> SHOW ALL RET
Domains:
        COMPANIES          DDMF_TEST          FAMILIES         FOOYAC
        OLD_FAMILIES       OWNERS             PROJECTS         YACHTS
        YACHTS_SEQUENTIAL
Records:
        COMPANIES_REC      FAMILY_REC         OWNER_RECORD     PROJECTS_REC
Procedures:
        LOA_REPORT         PRICE_PER_POUND    TMP              VERIFY
Tables:
The current dictionary is SY:[2,1]QUERY.DIC
No established collections
No ready domains
DTR>
```

You can print the definition of any dictionary object to which you have read access:

```
DTR> SHOW FAMILIES RET
DOMAIN FAMILIES
 USING FAMILY_REC ON FAMILY.DAT;
DTR>
```

The kind of access you have to a dictionary object depends on the access control list (ACL) associated with that object. The ACL specifies whether you have R (read), W (write), M (modify), E (execute), C (control) or no privilege for a dictionary object. You can only display the ACL for a dictionary object if you have C (control) privileges. The following example prints the ACL for the domains FAMILIES and PROJECTS:

```
DTR> SHOWP FAMILIES RET
        1,UIC, [*,*], "RWMEC"
DTR> SHOWP PROJECTS RET
        1,UIC, [23,45], "RWMEC"
        2,PW, "SESAME", "RE"
DTR>
```

See Chapter 19 for further information on controlling access to dictionary objects.

20.2 Modifying Dictionary Objects

To modify the definition of a dictionary object, you must have M (modify) access to the dictionary object. You can modify dictionary objects in the following ways:

- You can use the EDIT command to modify any dictionary object, using the EDT editing language.
- You can use the REDEFINE command to create a new definition. DATATRIEVE deletes the previous version of the object and allows you to define it a different way. The domain FAMILIES, for example, can be redefined as follows:

```
DTR> SHOW FAMILIES RET
DOMAIN FAMILIES
 USING FAMILY_REC ON FAMILY.DAT;
DTR> REDEFINE FAMILIES USING RET
DFN> NEW_FAMILY_REC ON NEW_FAMILY.DAT; RET
DTR> SHOW FAMILIES RET
DOMAIN FAMILIES
 USING NEW_FAMILY_REC ON NEW_FAMILY.DAT;
DTR>
```

NOTE

The previous definition of an element will be permanently lost after a REDEFINE command is entered. If you make a mistake while entering the definition, you must enter it again from the beginning.

- Another way to modify an object is to copy the definition to a command file using the EXTRACT command:

```
DTR> EXTRACT ON TEMP.CMD PERSON_REC RET
DTR>
```

Exit DATATRIEVE and edit the command file using the text editor of your choice. After making the needed changes, return to DATATRIEVE and execute the command file.

The EXTRACT command adds both a DELETE command and a DEFINE command to the beginning of the indirect command file. When you execute the command file, the DELETE command removes the old definition of the dictionary object from the current data dictionary, and the DEFINE command stores the new definition in that dictionary.

NOTE

Use extreme caution when changing record definitions. If you change the record definition so that the record it describes no longer matches the record stored in your file, you can no longer access your data. Chapter 15 explains the relationship between your record definition and data access.

The REDEFINE and EXTRACT commands do not copy the ACL associated with your record definition. When you redefine your record with the command file, DATATRIEVE also defines a new ACL for the record. This new ACL specifies the default privileges that have been set up for your system.

The QXTR utility, discussed later in this chapter, enables you to copy the ACL associated with your record to a command file.

20.3 Deleting Dictionary Objects

To remove the definition of a dictionary object from the current data dictionary, you must have C (control) access to the dictionary object. To remove a dictionary definition, use the DELETE command:

```
DTR> SHOW DOMAINS RET
Domains:
        DDMF_TEST       FAMILIES        FOOYAC          OLD_FAMILIES
        OWNERS          PROJECTS        YACHTS          YACHTS_SEQUENTIAL
DTR> DELETE FOOYAC; RET
DTR> SHOW DOMAINS RET
Domains:
        DDMF_TEST       FAMILIES        OLD_FAMILIES    OWNERS
        PROJECTS        YACHTS          YACHTS_SEQUENTIAL
DTR>
```

Remember to terminate the DELETE command with a semicolon.

DELETE removes from the dictionary both the definition of the dictionary object and its associated ACL. This command does not delete the data file associated with a domain. The data file still resides in the directory where it was stored. Therefore, you can delete a domain definition and redefine it to access the same data file.

20.4 Optimizing Disk Storage of Data Dictionaries with QCPRS

The data dictionary is an indexed file stored on a disk. When you add and delete definitions from a dictionary, that file accumulates unused areas of disk space. To reclaim this wasted disk space, run the utility program QCPRS. QCPRS compresses the contents of the dictionary, eliminating unused disk space. Compressing your dictionary can also improve DATATRIEVE performance.

To compress a data dictionary, you should first determine how many blocks of storage your dictionary occupies. The following example determines the size of the dictionary KELLER.DIC that resides in directory [100,120] on the system disk. The example uses the DCL DIRECTORY command:

```
$ DIRECTORY/FULL SY:[100,120]KELLER.DIC RET
```

The resulting display tells you the size (in blocks) of the dictionary. Later on, QCPRS prompts you for the number of blocks it should allocate for the compressed file. The number you supply should equal or exceed the number of blocks the dictionary currently uses.

On a RSTS/E system, you should rename the dictionary before invoking the QCPRS utility. The easiest way to do this is to change the file extension:

```
$ RENAME KELLER.DIC KELLER.BAK RET
$
```

Then you invoke QCPRS in response to the operating system prompt. The following example assumes that you are using DCL to invoke QCPRS, and that the QCPRS utility is resident in your current default directory. When invoked, QCPRS prints an identification message and displays a prompt, as follows:

```
$ RUN $QCPRS
QUERY FILE COPY-COMPRESS UTILITY

CPR>
```

Use the following format to compress the dictionary:

```
new-file = old-file
```

New-file is the file specification for the compressed copy of the dictionary. *Old-file* is the file specification of the dictionary to be compressed. If you omit a field in either file specification, QCPRS uses the following defaults:

Field	Default
dev:	SY: (the system device)
UIC/PPN	Your default UIC/PPN
file-name	QUERY
extension	DIC

Under all operating systems but RSTS/E, the file specifications for new-file and old-file can be the same. QCPRS merely creates a new copy of the file using the next higher version number.

Under RSTS/E, the file specifications for new-file and old-file must be different. Using the renamed dictionary from a previous example:

```
CPR> KELLER.DIC = KELLER.BAK RET
```

After you have entered the file specifications, QCPRS asks you to specify a number of disk blocks as an allocation for the new version of the dictionary:

```
ENTER ALLOCATION FOR AREA 0:
```

Enter the number of disk blocks you want to allocate for the compressed dictionary. If the number is too low, QCPRS automatically extends the file to hold the contents of the original file. If the number is too high, the extra blocks remain in the file and give room for contiguous expansion of the dictionary. An allocation greater than what the file currently needs can help maintain DATATRIEVE performance for a longer period of time; however, the higher number wastes disk space over the short run.

QCPRS then prompts again with CPR>, and you can compress another dictionary file or terminate QCPRS with CTRL/Z.

Because QCPRS does not alter the contents of the original file, you can save the file as a backup, or you can delete it.

You can use QCPRS to compress an indexed data file associated with a DATATRIEVE domain as well as to compress a dictionary. If you have added many records to the indexed data file since the time you created it, compressing the file can help reduce DATATRIEVE response time to queries that access the file. Simply follow the steps you use to compress a dictionary, but specify the name and extension of the indexed file.

20.5 Extracting Dictionary Content with the QXTR Utility

You might want to transfer dictionary objects from one data dictionary to another or transport your dictionary objects to VAX DATATRIEVE. You can use the QXTR utility to create a command file containing all the definitions extracted from a DATATRIEVE–11 data dictionary. Like QCPRS, QXTR is a dictionary maintenance program supplied with the DATATRIEVE–11 installation kit.

Running the QXTR program is equivalent to specifying all the objects in your dictionary in an EXTRACT command. However, the QXTR program also allows you to preserve the access control lists (ACLs) associated with each dictionary object.

You must invoke QXTR from the system command level. The following example uses the DCL RUN command:

```
$ RUN $QXTR RET
Extract Utility for DTR Dictionaries V02.00
```

QXTR then prompts you for the following information:

- The file specification of the dictionary to be processed.
- Whether you want to extract the access control list for each dictionary object.
- If you respond with Y to the question on access control lists, whether you want those lists to use VAX DATATRIEVE syntax.
- The name of the output command file to contain the extracted definitions. (The default is QXTR.CMD in your default directory.)

When QXTR finishes processing your dictionary, it returns you to system command level. The file QXTR creates an RMS sequential file you can invoke as an indirect command file for DATATRIEVE.

The following example processes the dictionary KELLER.DIC and copies the dictionary object definitions to LESLIE.CMD. The ACLs are extracted along with the objects. The ACLs remain in DATATRIEVE–11 syntax:

```
Dictionary Filespec to Extract from? KELLER.DIC RET
Should Protection Tables be Extracted (Y or N)? Y RET
Extract in VAX-11 DATATRIEVE Syntax? N RET
Filespec to Extract elements to? LESLIE.CMD RET
$
```

Like the EXTRACT command in DATATRIEVE, the QXTR utility precedes each dictionary object definition with a DELETE command. It also adds an ALLOCATION LEFT_RIGHT clause if no ALLOCATION clause is specified for record definitions.

QXTR checks that you have R (read) privilege for the objects in the dictionary before extracting them. If you do not, it prints a message that the objects were not extracted, and the program continues. If you are logged in under a privileged account (with a UIC/PPN [1,x]), you can extract everything regardless of the access control list. If the access control list allows only password access and not UIC/PPN access, you must run QXTR from a privileged account to extract the element. The program checks your current UIC against the access control list.

QXTR aborts if it encounters a corrupt dictionary object. In this case, the command file contains definitions extracted before QXTR encountered the corrupt object. It does not contain the definition of the corrupt object or the definitions that would follow. DATATRIEVE extracts definitions in the following order: domains, procedures, records, and tables. Definitions of specific objects within these four types are extracted in alphabetical order according to the name of the object. Examine the incomplete command file to determine which dictionary object is corrupt. You must delete the corrupt object from the dictionary before running QXTR again.

Appendix A

Name Recognition and Single Record Context

When you use a field name as a value expression and when you display, modify, or erase one or more records, DATATRIEVE determines exactly which record or records are the targets of the action you propose.

For each of these actions, DATATRIEVE must first determine the context within which the action occurs. The **context** is the set of conditions that govern the way DATATRIEVE recognizes field names and determines which records are the targets of DATATRIEVE statements. Understanding the way DATATRIEVE manages context is especially important when you begin nesting DATATRIEVE statements.

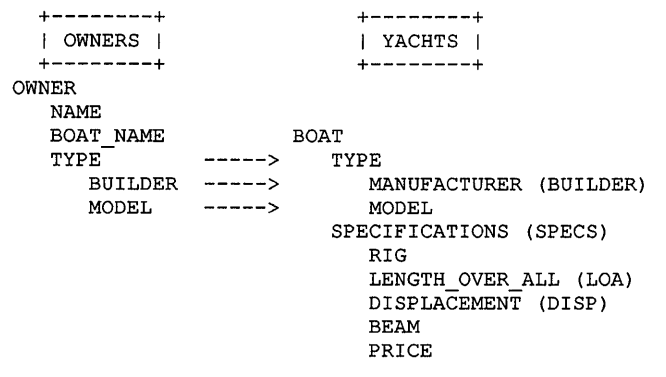
A.1 Establishing the Context for Name Recognition

DATATRIEVE does not require that every field name be unique. You can use the same name in several record definitions. You can even use the same name several times in the same record definition, as long as the fields with the identical name do not have the same level number in one group field.

Both the YACHTS and OWNERS domains, for example, have group fields named TYPE, and both group fields contain elementary fields you can refer to with the names BUILDER and MODEL. (In YACHTS, DATATRIEVE recognizes the query name BUILDER as equivalent to MANUFACTURER. Other query names for YACHTS are SPECS, LOA, and DISP.) Figure A–1 shows the fields in both record definitions and points out the duplicate names.

When you work with several record streams from the same domain, the field names in all record streams are identical. Whether you form collections or record streams of records from the YACHTS domain, DATATRIEVE has a mechanism for identifying which record to act on when you want to retrieve or change data from only one field of one record.

Figure A–1: Duplicate Field Names in YACHTS and OWNERS



When you understand the way DATATRIEVE establishes the context for recognizing names, you can use the names of domains, fields, collections, and variables to form both simple and complex relationships among fields. One of the keys to mastering the use of context is understanding the two DATATRIEVE context stacks.

A.1.1 The Right Context Stack

When you issue a statement, DATATRIEVE builds a **context stack**, a linked list that controls the DATATRIEVE search for names to match the ones you use in statements. The context stack consists of **context blocks**, or lists of names. These context blocks are linked together by pointers that control the sequence of search by DATATRIEVE for values to associate with the names you use in statements.

DATATRIEVE searches the right context stack for values to associate with names you use in print lists, Boolean expressions, and the right side of assignment statements such as x = y. The left context stack is discussed later in this appendix.

A.1.1.1 The Content of a Context Block

When you use a record selection expression, DATATRIEVE creates a context block to establish a context for name recognition. That context block contains, among other things, a list of names.

At the top of the list is a slot for the name of a context variable (see the section on context variables later in this appendix). Next is the name of the domain referred to in the record selection expression. The rest of the list contains the names of fields in the record associated with that domain. Those field names are arranged according to the field tree associated with the source.

The **field tree** contains the names of all the group fields, elementary fields, COMPUTED BY fields, REDEFINES fields, and lists in the record and preserves the hierarchical relationships among them.

When DATATRIEVE searches for a name in the context stack, it is looking for a value to associate with that name. The search ends, and DATATRIEVE takes the associated value when it finds the first name that matches the one in your statement.

A DATATRIEVE name can consist of several names joined together. They resemble dictionary path names in form and function. To be recognized, these compound or qualified names must represent a valid path through the hierarchy of a context block and the field tree it contains.

When DATATRIEVE encounters a name, it begins its search in the context block on top of the stack. DATATRIEVE first looks at the slot in the context block reserved for the name of a context variable. For unnamed CURRENT collections, this slot contains the name CURRENT. For named CURRENT collections, the name CURRENT and the collection name are equivalent. Named collections that are not the CURRENT collection have the collection name in this slot.

If the top block on the context stack refers to a record stream, this slot is empty unless you use a context variable in the RSE that forms the record stream. The context variable gives a record stream a temporary name; this name fills the first slot in the context block for these "named" record streams.

If DATATRIEVE finds that the first segment of a qualified name matches the name in the collection name/context variable slot, it continues its search in that block for a match for the rest of the name. If the name in your statement does not match the name in the collection name/context variable slot, or if that slot is empty, DATATRIEVE continues to look through the first context block to find a match.

Next in the context block is the name of the source of the records referred to by that block. For collections and record streams, that source can be the name of a domain, collection, or list for hierarchical records. The source can also be the name of a collection if you use the collection as the basis for a record stream in a FOR statement and you use a context variable.

If the source name does not match the name in your statement, DATATRIEVE next looks for the name in the slot reserved for names.

Next DATATRIEVE looks at the name of the top-level (the 01 level) field name. If no match occurs, DATATRIEVE looks at each succeeding field name in the order they are displayed when you enter a SHOW FIELDS command. That order can take you through the entire hierarchy of the field tree, traversing first the left branch then the right, wherever there is a branching point in the hierarchy.

If DATATRIEVE finds no match in the first block on the context stack, it goes to the next context block on the stack and begins its search there.

DATATRIEVE stops its search as soon as it finds an exact match for the name in your statement. Then it associates the value assigned to the name on the context stack with the name of the field in your statement.

If DATATRIEVE finds no match for the name in any of the context blocks, it displays a message on your terminal that the field name is either undefined or used out of context. The only remedies are to change the context so that the name in your statement resolves properly or to remove any ambiguity by qualifying the name further with group field names or context variables.

For the sake of clarity, the following description of the various types of context blocks starts with the bottom of the context stack, that is, with the context block that DATATRIEVE checks last.

A.1.1.2 Global Variables

The bottom context block contains the names of any global variables you have established and have not released. This block is different from the others on the stack because its content is not determined by a record selection expression. Nevertheless, DATATRIEVE treats the name of a global variable as though it were the name of a field in a simple record. Just as DATATRIEVE associates the value of a field with the field name, DATATRIEVE associates the value of a global variable with its name.

DATATRIEVE looks at the global variables last when trying to find a name to match one in your statement. No two global variables can have the same name. When you issue a DECLARE statement at command level (indicated by the DTR> prompt), DATATRIEVE checks the names of the global variables you have declared. If it finds one with the same name, it releases the old variable and its value and replaces it with the new one. DATATRIEVE initializes the new variable with a default value, a zero, or a space depending on the clauses you include in the DECLARE statement.

A.1.1.3 Collections

The next higher set of blocks in the context stack refers to existing collections. Each collection with a block on the context stack must have one record singled out as a selected record. Although a collection can have a number of records in it, only one of those records can be used in the search for the context of a name. DATATRIEVE can assign only one value to the name. Consequently, that one value can come from only one of the records in the collection.

Remember, the reason for resolving the context of a name you use in a statement is to assign a value to the name that can be used in the statement.

For an existing collection, you can designate one record at a time as the selected record for that collection. The SELECT statement lets you designate the selected record in a collection by relative reference (FIRST, NEXT, and LAST) or by absolute reference to the position number of the record in the collection. A collection has a block on the context stack only if it has a selected record.

If you have more than one existing collection with a selected record, the block immediately above the one for global variables refers to a named collection with a selected record. That collection is the one you formed with a FIND statement before you formed any of the other collections that have selected records.

The rest of the context blocks for the collections with selected records are ordered according to the sequence in which you formed them, not the order in which you entered the SELECT statement to establish the selected records.

If the CURRENT collection has a selected record, the context stack contains a block referring to the CURRENT collection. That block is above the blocks of all other collections. DATATRIEVE searches for names in the context block of the CURRENT collection before it searches the context block of any other collection.

The key to understanding the way DATATRIEVE recognizes names is that except for the global variables, the context stack is ordered on a last-in, first-out basis. The most recently formed context block is the one DATATRIEVE searches first.

You do not have to rely on your memory to recall the order in which you formed your existing collections. You need only issue a SHOW COLLECTIONS command. DATATRIEVE displays the most recently formed collection (always the CURRENT collection, whether it has a name or not) at the top of the list and the "oldest" one at the bottom.

The SHOW COLLECTIONS command, however, lists all the existing collections whether or not they contain selected records. Remember, only the collections with selected records are represented on the context stack.

With the SHOW collection-name command, you can inspect each existing collection to see how many records are in the collection, whether it has a selected record, and, if it does, what the position number of the selected record is in the collection.

If DATATRIEVE searches the context stack and does not find a match for the name in your statement, it displays an error message that may seem puzzling unless you understand the way DATATRIEVE forms the context stack:

```
Field "name" is undefined or used out of context
```

You may know that the name has been defined, and that it is the name of a field in a record associated with one or more existing collections. If, however, none of the collections containing that field has selected records, DATATRIEVE cannot tell if the field is defined or not.

If a collection containing the named field has no selected record, that collection has no block on the context stack. Consequently, DATATRIEVE neither finds a match for the field name nor has a way of discovering from the search of the context stack if the field name is defined at all.

The order of context blocks at the higher levels of the context stack depends on the order in which DATATRIEVE encounters the elements containing names associated with values. The order of the following sections does not imply any relative position on the stack. Only the order DATATRIEVE encounters those elements determines their order on the stack.

A.1.1.4 Record Streams

Before DATATRIEVE looks at the context block of the most recently formed collection with a selected record, it first looks at the context blocks created explicitly in the statement. One type of context block created by a statement refers to the field names of a record stream formed by a statement.

Context blocks of record streams act differently from those of collections. The context block for a collection stays on the stack as long as the collection has a selected record. The context block of a collection is removed from the stack only if you release the collection or remove its selected record with a DROP statement.

The context block for a record stream, however, stays on the stack only as long as the statement containing it is being executed. When DATATRIEVE finishes processing the statement, the block is removed from the context stack and is not available when DATATRIEVE rebuilds the stack after it encounters the next statement.

Only three statements and one command make lasting changes to the context stack:

- FIND

  The FIND statement can remove the CURRENT collection from the context stack by forming a new CURRENT collection. The new CURRENT collection releases the old collection but does not put a block on the context stack because a newly formed collection has no selected record.

- SELECT

  The SELECT statement puts a collection on the context stack by establishing a selected record. SELECT cannot change the relative order of collections on the stack. That order is determined by the relative order in which you formed the collections with the FIND statement.

- DROP

  The DROP statement removes a collection from the context stack by dropping the selected record from the collection. The SHOW collection-name command still notes the position number of the previously selected record, but the record has been removed from the collection and you cannot retrieve it unless you form a new collection that contains it.

- RELEASE

  The RELEASE command also removes a collection from the context stack. The released collection no longer exists, thus freeing the space it occupied. Records and domains associated with a collection named in the RELEASE command are not affected.

These three statements, however, share a restriction that separates them from all other statements: you cannot use FIND, SELECT, or DROP statements in compound statements. They must be entered at command level by themselves. Furthermore, these statements do not form temporary record streams; they affect only collections.

You can, however, have several context blocks for record streams on the context stack at one time. The block for a record stream stays on the context stack until DATATRIEVE finishes the statement. Because you can nest statements in FOR loops, BEGIN-END blocks, IF-THEN-ELSE statements, THEN, and WHILE statements, the inner statements can form record streams before DATATRIEVE finishes the outermost statement.

DATATRIEVE has to keep the context of outer statements separate from that of inner ones. It keeps them separate by putting a block on the context stack when it encounters an element that requires one. DATATRIEVE begins processing compound statements with the outermost statement and works progressively toward the innermost one. The context blocks it forms for elements in the innermost statement are at the top of the stack when the innermost statement is being processed.

When DATATRIEVE finishes processing the innermost statement, it removes the blocks created by that statement. DATATRIEVE works its way back out toward the outermost statement, removing blocks created by statements as soon as it finishes processing the statement. For example, in the case of nested FOR loops, the context block for the innermost FOR loop is higher in the stack than the blocks for the outer loops.

When DATATRIEVE completes the execution of the innermost loop, it removes the context block of that FOR statement, leaving the blocks of the outer FOR statement on the stack. As DATATRIEVE completes each loop, the context block for that loop is removed from the stack. This same pattern applies to statements in BEGIN-END blocks.

When a statement that forms a record stream is followed by a second statement that is not contained in the first, DATATRIEVE removes the context block created for the first statement from the stack and puts a context block for the second statement in its place. For example, in a BEGIN-END block, one PRINT statement containing an OF rse clause follows another. The context block of the first statement is in effect only during the execution of that first statement. That block is replaced by the one for the second PRINT statement when DATATRIEVE begins processing the second statement.

DATATRIEVE handles the context block of a FOR loop the same as it handles statements containing an OF rse clause.

DATATRIEVE creates four other types of context blocks that affect the order of the context stack: those for local variables, VERIFY clauses, VALID IF clauses, and context variables.

A.1.1.5 Local Variables

Local variables are variables defined in compound statements. A local variable and its effect on the context stack last only from the DECLARE statement that defines it until DATATRIEVE completes the execution of the statement containing the DECLARE statement.

A.1.1.6 VERIFY Clause in the STORE Statement

Like the context for local variables, the context for resolving field names in a VERIFY clause of the STORE statement is short-lived. The STORE statement does not access or change any existing record. Consequently, for each STORE statement, DATATRIEVE creates a context block to associate the field names with the values in the new record. DATATRIEVE executes the VERIFY clause after you have assigned values to all the fields prescribed by the syntax of the statement but before DATATRIEVE stores the record in the data file.

A.1.1.7 VALID IF Clause in a Record Definition

When you assign a value to a field name in either a STORE or MODIFY statement, DATATRIEVE looks in the appropriate record definition for a VALID IF clause. If the value is unacceptable according to the conditions specified in the VALID IF clause, DATATRIEVE displays a message on your terminal and reprompts you for an acceptable value. It uses the same context to associate the field name with your response to the reprompt.

The context for resolving field names in the VALID IF clause is established by the context block set up for either:

- The STORE statement
- The MODIFY statement

In either case, the value associated with the field name is the one just assigned to it by your response to a prompt or by an assignment statement in the USING clause of the STORE or MODIFY statement.

DATATRIEVE executes the VERIFY clause only after the values you assign meet the conditions of VALID IF clauses in the record definition. As a result, there can be no conflict between the context established for these two clauses. The context for the VALID IF clause no longer exists when DATATRIEVE executes the VERIFY clause.

A.1.2 Using Context Variables and Qualified Field Names

The ways of establishing context discussed to this point deal with resolving the connections between names and values by finding the first instance of a valid field name or variable name. When several context blocks on the stack contain fields with the same names, you need a way to skip over some instances of the name to get to the field that contains the value you want to retrieve.

DATATRIEVE gives you two methods of forcing name recognition: context variables and qualified field names. Although they require different actions from you, these two methods have an underlying similarity.

A.1.2.1 Context Variables as Field Name Qualifiers

A **context variable** is a dummy variable specified in a record selection expression for the purpose of name recognition. When DATATRIEVE encounters a context variable, it puts a new block on the context stack. That new block connects the name of the context variable with the field names and values of the records identified by the record selection expression.

The context established by the context variable lasts until DATATRIEVE completes the execution of the statement containing the record selection expression in which the context variable occurs. However, that context does not affect any outer loops or nesting statements that contain the statement in which you use the context variable.

A context variable, however, does affect all inner statements nested in the statement that contains the record selection expression in which the context variable occurs.

You can use the context variable as a prefix for each field name of the records identified by the record selection expression. Citing a field name with a context variable prefix can make a field name unique, even when the domains and field trees of a record in a record stream are identical.

Putting a prefix on a field name produces a qualified field name. The context variable must be the first prefix added to a field name.

A.1.2.2 Other Field Name Qualifiers

Using other qualifiers as prefixes to field names is the second method of overriding the DATATRIEVE default mechanism of name recognition.

Each fully qualified field name must be unique. The fully qualified field name consists of the record name, the top-level group field name, the names of any group field to which the elementary field belongs, and the elementary field name. You must separate each element of the fully qualified name from the next with a period. For example, in the domain YACHTS, the fully qualified field name of MODEL is the following:

```
YACHT.BOAT.TYPE.MODEL
```

You can use these elements in any combination that preserves their hierarchical order to distinguish the MODEL field in YACHTS from the MODEL field in another domain, such as OWNERS.

When DATATRIEVE encounters a qualified field name, it searches the context stack for the first match of the name you specify. For example, if you use BOAT.MODEL in a record selection expression, DATATRIEVE searches the context stack for the first valid occurrence of the name BOAT and searches the branches of the hierarchy under BOAT for the first valid occurrence of the name MODEL.

The success of the search is not jeopardized because you omit the group field name TYPE from the qualified name MODEL. DATATRIEVE searches the entire hierarchy under BOAT until it finds the first valid occurrence of TYPE. When an intermediary group field name is omitted, DATATRIEVE searches the hierarchy according to the order in which the fields of the record were defined.

Fully qualified field names are adequate when working with two or more domains that share elementary or group field names, or both. However, when you are working with two record streams from the same domain, you must further qualify the field name with a context variable. This extra qualification is especially necessary when dealing with lists in hierarchical records.

Suppose you want to display information about all builders who build boats with more than one type of rig. YACHT is the given name of the record associated with the domain YACHTS. The field tree of YACHT has the following structure:

```
YACHTS
   01 BOAT
     03 TYPE
        06 MANUFACTURER
        06 MODEL
     03 SPECIFICATIONS
        06 RIG
        06 LENGTH_OVER_ALL
        06 DISPLACEMENT
        06 BEAM
        06 PRICE
```

You can print the desired information with nested FOR loops. For each boat from the outer FOR statement, you want DATATRIEVE to loop through all the boats and find all the ones with the same builder. For each one it finds, you want to compare its rig with the rig of the boat from the outer loop. Then you want to separate out the ones for which the rigs are not the same. At first, you might be tempted to use the following statement to produce the desired list:

```
DTR> SET NO PROMPT RET
DTR> FOR YACHTS RET
CON>    FOR YACHTS WITH BUILDER = BUILDER AND RET
CON>      RIG NE RIG RET
CON>      PRINT BUILDER, RIG, RIG RET
DTR>
```

After a long search for records, DATATRIEVE displays no records. The problem is that the preceding syntax asks DATATRIEVE to look for a boat with a rig that is not equal to itself—an obvious contradiction. Both of the fields named RIG resolve to the record stream formed by the second FOR statement. The name BUILDER also resolves to the same record stream.

When you enter this statement, DATATRIEVE takes the first record from YACHTS but does not look at any of the values in its fields. Then it looks at every record in YACHTS and discovers that for every one of them, the name of the builder equals itself, but that no rig is not equal to itself. Thus every record in YACHTS fails to meet the condition set by the statement.

DATATRIEVE then takes the second record in YACHTS and once again goes through all the boats, finding that the two values are always equal to themselves and thus fail to meet the impossible demands of the statement. And so it goes for each record: two comparisons for 113 times 113 records, and no records meet the self-contradictory conditions.

The problem is how to get DATATRIEVE to look at the builder and rig of the outer FOR statement when making the comparison. The context variable provides one solution:

```
DTR> FOR A IN YACHTS RET
CON>    FOR YACHTS WITH BUILDER = A.BUILDER AND RIG NE A.RIG RET
CON>          PRINT BUILDER, A.RIG, RIG RET

MANUFACTURER  RIG    RIG

 AMERICAN     SLOOP  MS
 AMERICAN     MS     SLOOP
 CHALLENGER   SLOOP  KETCH
    .          .      .
    .          .      .
    .          .      .
 PEARSON      KETCH  SLOOP
 PEARSON      KETCH  SLOOP

DTR>
```

In this case, the use of the context variable A forces DATATRIEVE to look to the record stream formed by the outer FOR statement. At the same time, DATATRIEVE recognizes the unqualified names, RIG and BUILDER, in the context established by the most recent RSE: the one in the second FOR statement. The conditions in the second FOR statement are no longer impossible, and information from 62 records is displayed.

The way DATATRIEVE treats the unqualified names in this example illustrates another rule for context resolution: the left-hand member of a Boolean expression must resolve to the record selection expression of which it is a part. If you start the Boolean expression in the second FOR statement with A.BUILDER, DATATRIEVE tells you that A.BUILDER is undefined or used out of context.

You can add a second context variable in the previous example to make sure the resolution of the names is explicitly stated:

```
DTR> FOR A IN YACHTS RET
CON>  FOR B IN YACHTS WITH B.BUILDER = A.BUILDER AND B.RIG NE A.RIG RET
CON>          PRINT B.BUILDER, A.RIG, B.RIG RET
```

You gain two advantages by specifying the second context variable: clarity of representation and certainty that DATATRIEVE will display an error message if you make a syntax error. Using the second context variable, however, does not allow you to violate the rule for resolving field names on the left side of Boolean expressions.

A.1.3 The Left Context Stack for Assignment Statements

When you make assignment statements at DATATRIEVE command level or as part of STORE or MODIFY statements, DATATRIEVE must assign values to the field or variable you intend. It uses the left context stack to associate the values you supply with the fields and variables you want the values assigned to. Blocks on the left context stack are for records and variables that you can update.

Whenever DATATRIEVE begins to process a statement, the left context stack contains the global variables you have declared and not released. Any local variables you declare in compound statements are also on the left context stack. The local variables are removed when the statement in which you declared them ends.

Local and global variables are on both stacks. Each type of variable has a value that can be assigned to a field or another variable; hence, they are on the right context stack. Both can be updated with new values you assign them; hence, they are on the left context stack.

Context blocks for a record you want to modify is also on both context stacks. The record has a value you can use in Boolean expressions and assignment statements. You can update that value in a MODIFY statement. Because a field is on both stacks at the same time, you can use the old value of the field to calculate the new value. You can use the following form of assignment statement:

```
DTR> MODIFY USING PRICE = PRICE * 1.1 RET
DTR>
```

DATATRIEVE retrieves the old value of PRICE associated with the name on the right context stack and multiplies the old PRICE by a constant. It then associates that value with the name PRICE on the left context stack and updates the value of the PRICE field.

When you enter a STORE statement, the only context block for the new record is on the left context stack. No record exists yet, and, of course, no values are associated with fields of a record. The fields can only receive values.

However, as soon as DATATRIEVE associates a value with a field, you can move that value to the right context stack and use it on the right side of assignment statements. You can make this shift before you finish assigning values to all the fields of the new record. In fact, you can use the values of new fields to calculate the values DATATRIEVE stores in other new fields in the same record.

To shift newly stored values to the right context stack, include a context variable with the domain name when you enter the STORE statement:

```
DTR> STORE A IN YACHTS USING . . .
```

Then in the USING clause, you use the context variable to qualify the names of any field whose value you want to use on the right side of an assignment statement:

```
DTR> STORE A IN YACHTS USING RET
CON> BEGIN RET
CON>     F1 = value-expression RET
CON>     F2 = value-expression RET
CON>     F3 = A.F1 + A.F2 RET
CON> END RET
DTR>
```

The context variable allows you to associate a field name on the right context stack with its new value as soon as you assign the value to the field. You cannot, however, use a field name on the right side of an assignment statement until you have assigned a value to the field.

A.1.4 Examples of Context Variables in STORE and MODIFY Statements

You can combine STORE and MODIFY statements to keep an audit trail of modifications made to records in a domain and to change statistical records when you store new records.

To form an audit trail you need a domain for the audit records. This domain can use the same record definition as the original domain, but it must have its own domain definition and its own data file. The following is a simple example:

```
DTR> SHOW AUDIT_YACHTS RET
DOMAIN AUDIT_YACHTS USING
  YACHT ON AUD_YACHT;
DTR> FOR A IN YACHTS MODIFY USING RET
CON> BEGIN RET
CON>     BUILDER = *.BUILDER RET
CON>     MODEL =   *.MODEL RET
CON>     RIG =     *.RIG RET
CON>     LOA =     *.LOA RET
CON>     DISP =    *.WEIGHT RET
CON>     BEAM =    *.BEAM RET
CON>     PRICE =   *.PRICE RET
CON>     STORE B IN AUDIT_YACHTS USING RET
CON>        B.BOAT = A.BOAT RET
CON> END RET
Enter BUILDER:
```

If you have a VERIFY USING clause in the MODIFY statement, put the STORE statement as the last statement in the VERIFY clause. If you put the VERIFY clause after the STORE statement and the VERIFY clause aborts the change, you have a record of the change, but you have not changed the record.

You can also embed a MODIFY statement in a STORE statement. In this example, the embedded MODIFY statement updates a record of the last date that a new record was added to the data file, and records the TYPE field of the record stored. The file LAST.DAT is a sequential file with one record in it.

```
DTR> SHOW LAST_ENTRY RET
DOMAIN LAST_ENTRY USING LAST_REC ON LAST.DAT;
DTR> SHOW LAST_REC RET
RECORD LAST_REC USING
01 TOP.
03 LAST_DATE USAGE DATE.
03 TYPE PIC X(20).
;
DTR> STORE YACHTS USING RET
CON> BEGIN RET
CON>     BUILDER = *.BUILDER RET
CON>     MODEL =   *.MODEL RET
CON>     RIG =     *.RIG RET
CON>     LOA =     *.LOA RET
CON>     DISP =    *.WEIGHT RET
CON>     BEAM =    *.BEAM RET
CON>     PRICE =   *.PRICE RET
CON>     MODIFY LAST_ENTRY USING RET
CON>        BEGIN
CON>           LAST_DATE = "TODAY" RET
CON>           B.TYPE = A.TYPE RET
CON>        END
CON> END RET
Enter BUILDER:
```

With the proper use of context variables, you can also store or change data in fields shared by two or more domains.

A.2 Single Record Context

The DATATRIEVE statements PRINT, MODIFY, and ERASE can act on one record at a time or on an entire record stream or collection. The records on which they act are called **target records**. You can identify target records for these statements in four ways:

- A SELECT statement identifies one target record in a collection.
- The keyword ALL in the statement without an OF rse clause makes all records in a collection the targets of the statement.
- An OF rse clause in the statement forms a target record stream.
- The RSE clause in a FOR statement forms a stream of target records for the statement contained in the FOR loop.

A.2.1 The SELECT Statement and the Single Record Context

Before discussing the SELECT statement and context, a short review of facts about collections is in order.

DATATRIEVE keeps a list of the collections you form with the FIND statement. The most recent one formed is always at the top of the list and is called the CURRENT collection. The only other collections on the list are the ones to which you assigned a name when you formed them. The next collection you form then becomes the new CURRENT collection. DATATRIEVE discards the old CURRENT collection unless you give it a name when you form it.

With the RELEASE command, you can remove a collection from that list. If you release the CURRENT collection, the next one on the list becomes the CURRENT collection.

No collection on this list, however, is represented by a block on the context stack unless you use the SELECT statement to single out one record in the collection. When you select a record in a collection, DATATRIEVE puts a block for that collection on the context stack. If every existing collection has a selected record, then DATATRIEVE keeps a block on the context stack for each of those collections.

The relative ages of the collections with selected records determine the order of context blocks for collections. The "oldest" collection with a selected record is nearest the bottom of the context stack. Because the CURRENT collection is always the "youngest," its context block, if it has one, is nearest the top.

This order of context blocks for collections establishes the order DATATRIEVE uses not only for recognizing field names as described previously, but also for identifying single target records. When you enter the most abbreviated forms of the PRINT, MODIFY, and ERASE statements, DATATRIEVE looks on the context stack for the first valid single record context to carry out the specified action. It looks for the youngest collection with a selected record and either prints the record, erases it, or changes it.

The following sequence of examples illustrates the effect of the SELECT and DROP statements on single record context and the subsequent actions of the PRINT, MODIFY, and ERASE statements.

Form a collection of records from the YACHTS domain, call it BIGGIES, select the third record as the target record and display it:

```
DTR> READY YACHTS WRITE RET
DTR> FIND BIGGIES IN YACHTS WITH LOA > 40 RET
[8 records found]
DTR> SELECT 3 RET
DTR> PRINT RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL    WEIGHT BEAM  PRICE

 GULFSTAR     41         KETCH   41     22,000  12  $41,350

DTR>
```

Store a new record in the YACHTS domain and form a collection that consists of that one record. Later you can modify and erase this record:

```
DTR> STORE YACHTS RET
Enter MANUFACTURER: HINKLEY RET
Enter MODEL: BERMUDA 40 RET
Enter RIG: YAWL RET
Enter LENGTH_OVER_ALL: 40 RET
Enter DISPLACEMENT: 20000 RET
Enter BEAM: 12 RET
Enter PRICE: 82,000 RET
DTR> FIND YACHTS WITH BUILDER = "HINKLEY" RET
[1 record found]
DTR>
```

You now have two collections, CURRENT (the younger) and BIGGIES (the older):

```
DTR> SHOW COLLECTIONS RET
Collections:
       CURRENT
       BIGGIES

DTR> SHOW CURRENT RET
Collection CURRENT
        Domain: YACHTS
        Number of Records: 1
        No Selected Record
DTR> SHOW BIGGIES RET
Collection BIGGIES
        Domain: YACHTS
        Number of Records: 8
        Selected Record: 3
DTR>
```

The CURRENT collection has no selected record, but BIGGIES still does. Consequently, when you type PRINT and press the RETURN key again, DATATRIEVE prints the record in the first valid single record context, that is, the selected record in BIGGIES:

```
DTR> PRINT RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL      RIG   ALL   WEIGHT BEAM  PRICE

 GULFSTAR     41          KETCH  41    22,000  12  $41,350

DTR>
```

When you type SELECT and press the RETURN key, DATATRIEVE selects the first and only record in the CURRENT collection. Now when you type PRINT and press the RETURN key, the single record context has changed. The selected record in the CURRENT collection is the target record of the PRINT statement:

```
DTR> SELECT RET
DTR> PRINT RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL      RIG   ALL   WEIGHT BEAM  PRICE

 HINKLEY      BERMUDA 40 YAWL   40    20,000  12  $82,000

DTR> SHOW CURRENT RET
Collection CURRENT
        Domain: YACHTS
        Number of Records: 1
        Selected Record: 1
DTR>
```

Modify the PRICE of the target record and display the result. The MODIFY and PRINT statements both act on the record in the first valid single record context, that is, the selected record in the CURRENT collection:

```
DTR> MODIFY PRICE RET
Enter PRICE: 75,000 RET
DTR> PRINT RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL      RIG   ALL   WEIGHT BEAM  PRICE

 HINKLEY      BERMUDA 40 YAWL   40    20,000  12  $75,000

DTR>
```

Type ERASE and press the RETURN key. The ERASE statement also acts on the record in the first valid single record context, and the record for the HINKLEY boat is removed from the data file YACHT.DAT. Even though you erase the only record in the collection, DATATRIEVE does not discard the collection. It takes note that you have erased the selected record and removes the context block for the CURRENT collection from the context stack. You can verify the change in single record context by typing PRINT and pressing RETURN. The selected record from BIGGIES is again in the first valid single record context:

```
DTR> ERASE RET
DTR> SHOW CURRENT RET
Collection CURRENT
        Domain: YACHTS
        Number of Records: 1
        Selected Record: 1
DTR> PRINT RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL    WEIGHT BEAM  PRICE

 GULFSTAR     41         KETCH   41     22,000  12  $41,350

DTR>
```

If you type MODIFY or ERASE and press the RETURN key, and no existing collection has a selected record, DATATRIEVE displays a message that there is no target record for the action you propose:

```
DTR> ERASE RET
No target record for ERASE.
DTR> MODIFY RET
No selected record for modify
DTR>
```

However, if you type PRINT and press the RETURN key, and no existing collection has a selected record, DATATRIEVE displays a message that there is no selected record and then prints out the whole collection:

```
DTR> FIND YACHTS WITH BUILDER = "ALBIN" RET
[3 records found]
DTR> PRINT RET
No record selected, printing whole collection

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL    WEIGHT BEAM  PRICE

 ALBIN        79          SLOOP   26      4,200  10  $17,900
 ALBIN        BALLAD      SLOOP   30      7,276  10  $27,500
 ALBIN        VEGA        SLOOP   27      5,070  08  $18,600

DTR>
```

You can change the single record context with the DROP statement. The DROP statement removes the selected record from a collection but does not erase the record from the data file. When you type DROP and press the RETURN key, and the CURRENT collection has no selected record, DATATRIEVE displays a message on your terminal:

```
DTR> FIND BIGGIES IN YACHTS WITH LOA > 40 RET
[8 records found]
DTR> DROP RET
No collection with selected record for DROP
DTR>
```

If the CURRENT collection has a selected record, the DROP statement removes that record from the collection when you type DROP and press the RETURN key. If other collections have selected records, you must specify the collection name in the DROP statement.

The CURRENT collection is BIGGIES. Select and display the first record in BIGGIES and form a new CURRENT collection of boats built by ALBIN:

```
DTR> SELECT; PRINT RET

                                  LENGTH
                                   OVER
MANUFACTURER    MODEL      RIG     ALL    WEIGHT BEAM  PRICE

 CHALLENGER  41          KETCH    41     26,700  13  $51,228

DTR> FIND YACHTS WITH BUILDER = "ALBIN" RET
[3 records found]
DTR>
```

Now select, display, and drop the first record of the CURRENT collection. Then enter a SHOW CURRENT command to see how DATATRIEVE records the results of your actions. The SELECT statement creates a single record context for the current collection, thus the target record of the PRINT statement is the selected record in the CURRENT collection, not in BIGGIES:

```
DTR> SELECT RET
DTR> PRINT RET

                                  LENGTH
                                   OVER
MANUFACTURER    MODEL      RIG     ALL    WEIGHT BEAM  PRICE

 ALBIN        79          SLOOP    26      4,200  10  $17,900

DTR> DROP RET
DTR> SHOW CURRENT RET
Collection CURRENT
        Domain: YACHTS
        Number of Records: 3
        Selected Record: 1
DTR>
```

When you drop a selected record from a collection, you change the single record context. The context block for that collection is removed from the context stack.

Consequently, when you type PRINT and press the RETURN key again, DATATRIEVE displays the selected record in BIGGIES, the record in the first valid single record context:

```
DTR> PRINT RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 CHALLENGER  41          KETCH   41    26,700  13  $51,228

DTR>
```

Like PRINT, MODIFY, and ERASE, the DROP statement acts on the record in the first valid single record context.

If you type PRINT and press RETURN when you have no valid single record context, DATATRIEVE displays the whole CURRENT collection because there is no selected record in either of the two existing collections. Because you dropped one record from the CURRENT collection, it contains only two records now:

```
DTR> PRINT RET
No record selected, printing whole collection

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 ALBIN         BALLAD    SLOOP   30     7,276  10  $27,500
 ALBIN         VEGA      SLOOP   27     5,070  08  $18,600

DTR>
```

To show that you have not erased the record dropped from the CURRENT collection, form and display a new CURRENT collection of boats by ALBIN:

```
DTR> FIND YACHTS WITH BUILDER = "ALBIN" RET
[3 records found]
DTR> PRINT ALL RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL   WEIGHT BEAM  PRICE

 ALBIN         79        SLOOP   26     4,200  10  $17,900
 ALBIN         BALLAD    SLOOP   30     7,276  10  $27,500
 ALBIN         VEGA      SLOOP   27     5,070  08  $18,600

DTR>
```

A.2.2 The CURRENT Collection as Target Record Stream

The preceding example shows the effect of the keyword ALL on a PRINT statement that does not contain an OF rse clause.

Although DATATRIEVE acts on only one record at a time, you can identify more than one record for a single DATATRIEVE statement to act on. With the keyword ALL, you can make every record in the CURRENT collection the target of a single PRINT, MODIFY, or ERASE statement. Such a statement, however, cannot also contain an OF rse clause.

If you have a CURRENT collection and type PRINT ALL and press the RETURN key, DATATRIEVE displays the whole CURRENT collection. If you have no CURRENT collection, DATATRIEVE displays a message on your terminal. To illustrate this effect, release all collections and enter the statement PRINT ALL:

```
DTR> SHOW COLLECTIONS RET
Collections:
       CURRENT
       BIGGIES

DTR> RELEASE CURRENT, BIGGIES RET
DTR> SHOW COLLECTIONS RET
No established collections
DTR> PRINT ALL RET
A current collection has not been established
DTR>
```

DATATRIEVE displays the same message on your terminal when you have no CURRENT collection and you enter an ERASE ALL or MODIFY ALL statement.

When you have a CURRENT collection and enter an ERASE ALL statement, DATATRIEVE removes every record in the CURRENT collection from the data file. Although frequently useful, this operation can jeopardize valuable data if you use it carelessly.

Note that if your collection contains many records and you mistakenly enter an ERASE ALL or MODIFY ALL statement, you can enter CTRL/C to prevent all the records in the CURRENT collection from being erased or changed. How many records get erased or changed under such circumstances depends on how quickly you enter CTRL/C, the processing load on your system, and the priority of your process.

The various forms of the MODIFY ALL statement change the data in each record of the CURRENT collection (see the *DATATRIEVE–11 Reference Manual*). Make a collection of the first three yachts with no listed price. Display the CURRENT collection, modify the PRICE to $30,000, display the results of the change, and change the price back to zero using a different form of the MODIFY ALL statement:

```
DTR> FIND FIRST 3 YACHTS WITH PRICE = 0 RET
[3 records found]
DTR> PRINT ALL RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL    WEIGHT BEAM  PRICE

 BLOCK I.       40        SLOOP  39     18,500  12
 BUCCANEER      270       SLOOP  27      5,000  08
 BUCCANEER      320       SLOOP  32     12,500  10

DTR> MODIFY ALL PRICE RET
Enter PRICE: 30,000 RET
DTR> PRINT ALL RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL    WEIGHT BEAM  PRICE

 BLOCK I.       40        SLOOP  39     18,500  12  $30,000
 BUCCANEER      270       SLOOP  27      5,000  08  $30,000
 BUCCANEER      320       SLOOP  32     12,500  10  $30,000

DTR> MODIFY ALL USING PRICE = 0; PRINT ALL RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL    WEIGHT BEAM  PRICE

 BLOCK I.       40        SLOOP  39     18,500  12
 BUCCANEER      270       SLOOP  27      5,000  08
 BUCCANEER      320       SLOOP  32     12,500  10

DTR>
```

A.2.3 The OF rse Clause and Target Record Streams

The OF rse clause in a PRINT, ERASE, or MODIFY statement lets you create a new context for that statement. The OF rse clause specifies a target record stream that overrides any context established for your existing collections. For each such clause, DATATRIEVE puts a new block on the context stack. When DATATRIEVE completes execution of the statement, it removes that block from the context stack.

The following example contrasts the effect of PRINT, PRINT ALL, and PRINT OF rse. (When the PRINT statement does not include a list of fields, you can omit the OF from the statement). The record selection expression here is FIRST 3 YACHTS WITH PRICE = 0. This RSE identifies a new target record stream for the PRINT statement that overrides the CURRENT collection as a target record stream. It also overrides the single record context of the selected record in the CURRENT collection:

```
DTR> FIND FIRST 3 YACHTS RET
[3 records found]
DTR> SELECT; PRINT RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL    WEIGHT BEAM  PRICE

 ALBERG       37 MK II   KETCH   37    20,000  12   $36,951

DTR> PRINT ALL RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL    WEIGHT BEAM  PRICE

 ALBERG       37 MK II   KETCH   37    20,000  12   $36,951
 ALBIN        79         SLOOP   26     4,200  10   $17,900
 ALBIN        BALLAD     SLOOP   30     7,276  10   $27,500

DTR> PRINT FIRST 3 YACHTS WITH PRICE = 0 RET

                                LENGTH
                                 OVER
MANUFACTURER    MODEL     RIG    ALL    WEIGHT BEAM  PRICE

 BLOCK I.     40         SLOOP   39    18,500  12
 BUCCANEER    270        SLOOP   27     5,000  08
 BUCCANEER    320        SLOOP   32    12,500  10

DTR>
```

To reduce the risk to your data, DATATRIEVE forces you to include both keywords ALL and OF when using the OF rse clause in MODIFY and ERASE statements. Although the results are not shown here, you must use MODIFY and ERASE statements similar to the following examples (but with your choice of record selection expressions). The record selection expression used in these statements is PHONES WITH DEPT = "32T":

```
DTR> MODIFY ALL OF PHONES WITH DEPT = "32T"

DTR> MODIFY ALL DEPT OF PHONES WITH DEPT = "32T"

DTR> MODIFY ALL USING DEPT = _*."NEW DEPT" OF PHONES WITH DEPT = "32T"

DTR> ERASE ALL OF PHONES WITH DEPT = "32T"
```

Unless you include an assignment statement in the USING clause of a MODIFY statement, DATATRIEVE prompts you once to supply a value for each elementary field specified or implied in the statement. After you respond to the last of the prompts, DATATRIEVE begins to change each of the records in the CURRENT collection to correspond to the values you supplied to the prompts. You can prevent any changes from taking effect by entering CTRL/Z when responding to any of the prompts.

A.2.4 FOR Statements and Target Record Streams

You can use FOR statements to create target record streams for the DATATRIEVE statements that use single-record context. Using FOR loops has an advantage over using target record streams formed by the OF rse clause and the target record stream formed of the CURRENT collection by the keyword ALL. The FOR statement lets you work with each record individually; you do not have to perform the same operation on all target records. By putting STORE and MODIFY statements and prompting value expressions in a FOR loop, you can act on each member of a record stream or collection one at a time.

When you put a MODIFY statement in a FOR statement, DATATRIEVE prompts you once for each field in the record if you do not specify a field list or a USING clause in the MODIFY statement.

This FOR statement creates a record stream of boats that have no price listed. The MODIFY statement prompts you to supply a price for each record in the record stream. You can put a unique value in the PRICE field for each boat:

```
DTR> READY YACHTS MODIFY RET
DTR> FOR YACHTS WITH PRICE = 0 MODIFY PRICE RET
Enter PRICE: 12900 RET
Enter PRICE: 15600 RET
Enter PRICE:
                          .
                          .
                          .
DTR>
```

Another valuable feature of FOR loops is the complex relationships you create between record streams when you include one FOR loop inside another. Each FOR statement puts a block on the context stack. As a result, you can use the context mechanism to transfer values between records.

By putting a MODIFY statement inside two FOR statements, you can automatically update master records with the data from periodic transaction records:

```
DTR> FOR A IN DAILY_TRANSACTIONS RET
CON>   FOR B IN MASTER_DATA WITH B.ACCOUNT = A.ACCOUNT RET
CON>     MODIFY USING RET
CON>       BEGIN RET
CON>         MASTER_BAL    = MASTER_BAL    - WITHDRAW + DEPOSIT RET
CON>         TOT_WITHDRAW = TOT_WITHDRAW + WITHDRAW RET
CON>         TOT_DEPOSIT   = TOT_DEPOSIT   + DEPOSIT RET
CON>       END RET
DTR>
```

The Boolean expression in this example limits the record stream for the inner FOR statement to one record.

You can also create nested FOR statements in which DATATRIEVE executes a series of statements at each level of nesting. For each owner record in the next example, DATATRIEVE asks you if you want to modify the SPECS of every boat in the YACHTS inventory built by the manufacturer of the owner's boat. The third time through the outer loop, DATATRIEVE again begins the cycle of prompting for the boats by Albin because the third person in the OWNERS domain also owns a boat by Albin. Notice that the record changed during the second loop appears during the third:

```
DTR> SET NO PROMPT RET
DTR> FOR OWNERS RET
CON> BEGIN RET
CON>     PRINT SKIP, BUILDER, SKIP RET
CON>     FOR YACHTS WITH BOAT.BUILDER = OWNER.BUILDER RET
CON>         BEGIN RET
CON>            PRINT SPECS RET
CON>            IF *."DO YOU WANT TO CHANGE THIS" CONT "Y" RET
CON>               THEN MODIFY SPECS RET
CON>         END RET
CON> END

 BUILDER

ALBERG

        LENGTH
         OVER
 RIG     ALL    WEIGHT BEAM  PRICE

KETCH    37     20,000  12  $36,000
Enter DO YOU WANT TO CHANGE THIS: N RET

ALBIN

SLOOP    26      4,200  10  $17,900
Enter DO YOU WANT TO CHANGE THIS: N RET
SLOOP    30      7,276  10  $27,500
Enter DO YOU WANT TO CHANGE THIS: N RET
SLOOP    27      5,070  08  $18,600
Enter DO YOU WANT TO CHANGE THIS: Y RET
Enter RIG: KETCH RET
Enter LENGTH_OVER_ALL: 35 RET
Enter DISPLACEMENT: 17000 RET
Enter BEAM: 12 RET
Enter PRICE: 33000 RET
```

```
ALBIN

SLOOP   26     4,200  10  $17,900
Enter DO YOU WANT TO CHANGE THIS: N RET
SLOOP   30     7,276  10  $27,500
Enter DO YOU WANT TO CHANGE THIS: N RET
KETCH   35    17,000  12  $33,000
Enter DO YOU WANT TO CHANGE THIS: N RET

C&C

SLOOP   31     8,650  09
Enter DO YOU WANT TO CHANGE THIS: CTRL/Z
Execution terminated by operator
DTR>
```

Index

A

B

C

D

E

F

G

H

I

K

L

R

S

T

U

V

W

Y

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-DEC-DEMO (800-332-3366) using a 1200- or 2400-baud modem. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local DIGITAL subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	————	Local DIGITAL subsidiary or approved distributor
Internal[1]	————	SDC Order Processing - WMO/E15 *or* Software Distribution Center Digital Equipment Corporation Westminster, Massachusetts 01473

[1]For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

DATATRIEVE–11
User's Guide
AA–X023C–TK

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

What I like best about this manual is _____

_____

What I like least about this manual is _____

_____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35
110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

Do Not Tear - Fold Here

Reader's Comments

DATATRIEVE–11
User's Guide
AA–X023C–TK

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

What I like best about this manual is _____

_____

What I like least about this manual is _____

_____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

— Do Not Tear - Fold Here and Tape ----------

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01–3/J35
110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

— Do Not Tear - Fold Here ----------

digital™

Printed in U.S.A.