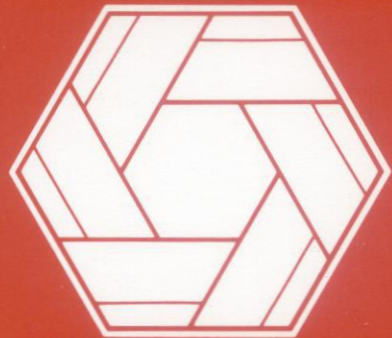


VAX DATATRIEVE

Handbook

Order No. AA-W675B-TE



digitalTM
software

**VAX DATATRIEVE
Handbook**

Order No. AA-W675B-TE

November 1987

This manual contains general information on using
VAX DATATRIEVE.

OPERATING SYSTEM:	VMS MicroVMS
SOFTWARE VERSION:	VAX DATATRIEVE V4.1

digital equipment corporation, maynard, massachusetts

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

Copyright © 1984, 1985, 1987 by Digital Equipment Corporation. All Rights Reserved.

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

The following are trademarks of Digital Equipment Corporation:

ACMS	PDP	VAX
CDD	RALLY	VAXcluster
DATATRIEVE	Rdb/ELN	VAXinfo
DEC	Rdb/VMS	VAX Information Architecture
DECnet	ReGIS	VAX/VMS
DECUS	TDMS	VIDA
MicroVAX	TEAMDATA	VMS
MicroVMS	UNIBUS	VT

digital™

IBM® is a registered trademark of International Business Machines Corporation.

How to Use This Manual	xiii
-------------------------------	------

Technical Changes and New Features	xix
---	-----

Part I Getting Started with DATATRIEVE

1 Getting Started with VAX DATATRIEVE

1.1 What Is DATATRIEVE?	1-1
1.2 Create and Use a VMS Directory.	1-2
1.3 Obtain a CDD Directory and Start DATATRIEVE.	1-6
1.4 Look at Some Sample Definitions and Data	1-9
1.5 End Data Access and Exit DATATRIEVE.	1-14
1.6 Create a Dictionary Subdirectory	1-15
1.7 Create a Simple Application	1-16
1.7.1 Use ADT to Create Data Definitions and a File.	1-17
1.7.2 Store Records	1-20
1.7.3 Display Data	1-21
1.7.4 Change Field Values	1-22
1.7.5 Change Domain Structure.	1-23
1.7.6 Create a Table	1-24
1.7.7 Write a Procedure	1-27
1.8 What Do I Read Next?	1-29
1.9 What DATATRIEVE Can Do for the Programmer	1-29

2 VMS Concepts

2.1 Using DIGITAL Command Language (DCL)	2-1
2.2 Logging In	2-2
2.2.1 Getting the Terminal Ready	2-2
2.2.2 Gaining Access to the System.	2-3
2.3 Getting Online Help	2-4
2.4 Entering Commands.	2-4
2.4.1 Command Prompting.	2-5
2.4.2 Defaults.	2-6
2.4.3 Abbreviating Commands.	2-6
2.4.4 Recovering from Errors	2-6
2.4.5 Summary of Entering Commands	2-7
2.5 Interpreting System Responses	2-8
2.5.1 Information Messages	2-8
2.5.2 Error Messages	2-8

2.6	Logging Out	2-9
2.7	File Management	2-9
2.7.1	Creating Files	2-10
2.7.2	Identifying Files	2-10
2.7.2.1	Nodes	2-10
2.7.2.2	Devices	2-11
2.7.2.3	Directories and Subdirectories	2-11
2.7.2.4	File Names, Types, and Versions	2-11
2.7.2.5	Wildcard Character	2-12
2.7.3	Deleting Files	2-13
2.7.4	Purging Files	2-13
2.7.5	Displaying Files at Your Terminal	2-14
2.7.6	Printing Files	2-14
2.7.7	Listing Files in a Directory	2-15
2.7.8	Copying Files	2-15
2.7.9	Renaming Files	2-16
2.7.10	Appending Files	2-17
2.7.11	Finding Differences Between Files	2-17
2.7.12	Searching Files for a Selected String	2-17
2.8	Creating and Managing VMS Directories	2-18
2.8.1	Directory Structure	2-18
2.8.2	Accessing Other Directories	2-19
2.8.3	Creating Subdirectories	2-20
2.8.4	Changing Your Default Directory	2-20
2.8.5	Protecting Your Directories and Files	2-21
2.8.6	Deleting a Directory	2-23
2.9	Logical Names	2-23
2.10	System Default Logical Names	2-24
2.11	Symbols	2-26
2.12	Command Procedures	2-27
2.12.1	A LOGIN.COM File	2-28
2.13	Finding More Information	2-28

3 Using Editors Within DATATRIEVE

3.1	General Editing Information	3-2
3.1.1	Assigning a DATATRIEVE Editor	3-2
3.1.2	Using Line Recall Within DATATRIEVE	3-3

3.1.3	Using the DATATRIEVE EDIT command	3-5
3.1.3.1	Editing the Previous DATATRIEVE Command or Statement	3-5
3.1.3.2	Privileges Needed to Edit CDD Objects	3-7
3.1.3.3	Editing a CDD Object Specified by Path Name	3-7
3.1.3.4	Editing by Types of Objects Within DATATRIEVE	3-7
3.1.3.5	Using EDIT to Recover from a System Failure	3-8
3.2	Using EDT Within DATATRIEVE	3-9
3.3	Using VAXTPU Within DATATRIEVE	3-11
3.4	Using LSE within DATATRIEVE.	3-13
3.5	Ending Your Editing Session.	3-16
3.5.1	Ending an EDT Session	3-16
3.5.2	Ending a VAXTPU Session	3-17
3.5.3	Ending an LSE Session.	3-17

4 Using Syntax Diagrams

4.1	Uppercase Words in Syntax Diagrams	4-4
4.2	Lowercase Words in Syntax Diagrams	4-4
4.3	Brackets and Braces in Syntax Diagrams	4-4
4.4	Separators in Syntax Diagrams	4-5

Part II Controlling your DATATRIEVE Environment

5 Input Options During a DATATRIEVE Session

5.1	Invoking DATATRIEVE	5-1
5.2	Creating a Startup Command File (DTR\$STARTUP).	5-2
5.3	Exiting DATATRIEVE	5-3
5.4	Getting DATATRIEVE to Process More Than One Line as a Unit.	5-4
5.4.1	Turning Off the "Looking for..." Messages	5-5
5.4.2	Using Standard Programming Conventions to Format Input	5-5

6 Getting Online Assistance

6.1	Using Help.	6-1
6.2	Getting Help on Errors	6-2
6.3	Guide Mode	6-3

7 Using the VAX Common Data Dictionary

7.1	What Is the CDD?	7-1
7.2	How Is the CDD Organized?	7-2

7.3	Creating and Using Path Names	7-4
7.3.1	Using Full Path Names	7-5
7.3.2	Version Numbers	7-6
7.3.3	Abbreviating Path Names	7-7
7.3.4	The Logical Name in Dictionary Path Names.	7-7
7.4	Setting Dictionary Location	7-9
7.5	Displaying Information About Directories, Objects, and Session Defaults	7-9
7.6	Deleting, Purging, and Extracting Definitions	7-12
7.7	Creating Dictionary Directories	7-14
7.8	Deleting Dictionary Directories	7-15
7.9	Displaying and Setting Protection for Directories and Objects	7-16
7.9.1	Displaying Your Privileges.	7-18
7.9.2	Displaying and Changing an ACL.	7-21
7.10	Using the CDD to Design Department-Wide or System-Wide Applications	7-23

Part III Setting Up a Database

8 Application Case Study: a Personnel System

8.1	Reviewing the Requirements	8-1
8.2	Analyzing the Data	8-4
8.3	Grouping Fields into Domains and Tables	8-6

9 Defining Records

9.1	Setting Up Relationships Among Fields (Level Numbers).	9-3
9.2	Selecting Names	9-5
9.2.1	Differences Between Record Name and Top-Level Field	9-6
9.2.2	Query Names.	9-7
9.2.3	Column Headers.	9-7
9.2.4	FILLER Field Name	9-8
9.3	Specifying Type and Size of Data	9-9
9.3.1	Specifying a PIC Clause	9-9
9.3.1.1	Defining Alphanumeric (X) and Alphabetic (A) Fields	9-10
9.3.1.2	Defining Numeric Fields	9-11
9.3.2	Specifying a USAGE Clause.	9-12
9.3.3	Date Fields	9-15
9.3.4	Virtual (Computed) Fields	9-15
9.3.5	Defining One Record Area in Different Ways Using the REDEFINES Clause.	9-16
9.3.6	Specifying Repeating Fields Using the OCCURS Clause.	9-17
9.4	Formatting the Display of Field Values	9-20

9.5	Including Validation Requirements	9-23
9.6	Initializing Field Values	9-23
9.7	Specifying Values to Be Ignored in Statistical Computations	9-23
9.8	Ending Field and Record Definitions	9-24
9.9	Editing Record Definitions	9-24

10 Defining Domains

10.1	Naming the Domain	10-2
10.2	Specifying the Record Name	10-3
10.3	Specifying the Data File	10-3
10.3.1	How Much of the File Specification to Include.	10-3
10.3.2	Avoiding Problems When Naming Files	10-4

11 Defining Data Files

11.1	Defining Indexed Files	11-2
11.1.1	Selecting the Primary Key	11-3
11.1.2	Selecting Alternate Keys	11-4
11.1.3	Selecting Group Field Keys	11-4
11.2	Defining Sequential Files	11-5
11.3	Planning for File Maintenance	11-6
11.3.1	Using Other Options in the DEFINE FILE Command	11-6
11.3.2	Using RMS Utilities to Load and Maintain Files	11-7
11.4	Restructuring a Domain	11-7
11.4.1	Changing Only File Organization, Storage Options, and Keys	11-7
11.4.2	Changing the Fields Defined in the Record Definition.	11-9
11.4.3	Restructuring a Domain to Add Its Records to Another Domain	11-12

12 Defining Tables

12.1	Creating Dictionary Tables	12-2
12.2	Creating Domain Tables	12-3
12.3	Using DATATRIEVE Tables	12-4
12.3.1	Access Privileges Needed to Use Tables	12-5
12.3.2	Accessing Values in Tables	12-5
12.3.3	Editing Table Definitions	12-7
12.3.4	Validating Values with Tables	12-8
12.4	Choosing Between Dictionary and Domain Tables	12-8

Part IV Data Retrieval and Maintenance

13 Starting and Ending Access to Data

- 13.1 Readyng Domains 13-3
 - 13.1.1 Defining Your Own Default Access 13-5
- 13.2 Finishing Domains 13-6

14 Retrieving Data the Easy Way: With Collections

- 14.1 Specifying the Records You Want in a Collection 14-4
- 14.2 Forming and Naming Collections 14-6
- 14.3 Choosing a Target Record for an Operation. 14-7
- 14.4 Restricting Record Fields to the Ones You Need. 14-10
- 14.5 Sorting Records 14-12
- 14.6 Forming a Collection from Two or More Record Sources 14-14
- 14.7 Removing Records from a Collection 14-15
- 14.8 Removing Collections from Your Workspace. 14-16
- 14.9 Disadvantages of Using Collections 14-16

15 Accessing Data the Expert Way: Without Collections

- 15.1 Processing Records from Domains Rather Than Collections 15-1
 - 15.1.1 Ensuring Fast Access 15-1
 - 15.1.2 Processing Records in Compound Statements 15-4
- 15.2 Creating RSEs 15-7
- 15.3 Working with Multiple Records. 15-9
 - 15.3.1 FOR Statement Looping Errors. 15-10
 - 15.3.2 CROSS Clause Looping Errors 15-10
 - 15.3.3 Lists, the "Record" Within the Record 15-11
- 15.4 Creating Views 15-14
 - 15.4.1 View Domains That Subset Fields from One Domain 15-14
 - 15.4.2 View Domains That Combine Fields from Two or More Domains 15-17
- 15.5 Access Privileges Needed for Using Views 15-19
- 15.6 Summary of Options: Advantages and Disadvantages 15-20

16 Maintaining Data

- 16.1 Storing Records 16-1
- 16.2 Erasing Records. 16-5
- 16.3 Modifying Records 16-10

Part V Programming with DATATRIEVE

17 Using Procedures and Compound Statements

17.1	Creating and Executing Procedures	17-1
17.2	Correcting and Changing Procedures	17-4
17.3	Using Compound Statements.	17-5
17.3.1	Combining Statements with the REPEAT Statement	17-7
17.3.2	Combining Statements with the FOR Statement	17-7
17.3.3	Combining Statements with the Keyword THEN	17-8
17.3.4	Combining Statements in a BEGIN-END Block	17-8
17.3.5	Combining Statements with the WHILE Statement	17-9
17.3.6	Combining Statements with the IF-THEN and IF-THEN-ELSE Statements	17-10
17.3.7	Combining Statements with the CHOICE Statement.	17-11
17.4	Guidelines for Writing Procedures and Compound Statements.	17-12
17.4.1	Using FIND, SELECT, SORT, REDUCE, and DROP Statements.	17-13
17.4.2	Avoiding Looping Mistakes	17-13
17.4.3	Invoking DATATRIEVE Procedures from VMS Command Files	17-14
17.4.4	Controlling Execution on Error Conditions	17-14
17.5	Getting a Procedure to Work the Way You Want	17-16
17.5.1	Displaying Command File and Procedure Input During Execution.	17-16
17.5.2	Writing a Session Log to a File.	17-18
17.5.3	Checking the Last Word or Character of Input Lines	17-18

18 Defining and Calculating Values with DATATRIEVE

18.1	Using DATATRIEVE Expressions	18-1
18.1.1	Value Expressions	18-2
18.1.1.1	Literals	18-3
18.1.1.2	Record Field Names	18-4
18.1.1.3	Variable Field Names.	18-4
18.1.1.4	Prompting Value Expressions	18-6
18.1.1.5	Arithmetic Expressions	18-7
18.1.1.6	Conditional Value Expressions	18-8
18.1.1.7	FORMAT Value Expressions.	18-10
18.1.2	Boolean Expressions	18-10
18.1.2.1	Relational Operators	18-11
18.1.2.2	Boolean Operators	18-14

18.2	Computing Sums and Other Statistics	18-16
18.2.1	Statistical Value Expressions	18-16
18.2.2	Using the SUM Statement and Statistical Value Expressions with the CURRENT Collection	18-20
18.3	Storing and Displaying Date and Time	18-22
18.3.1	Storing and Displaying Values in Date Fields	18-23
18.3.2	Comparing and Searching for Date Values	18-26
18.3.3	Subtracting Values from a Date Field.	18-28

Part VI Formatting Displays and Writing Reports

19 Improving Screen Displays and Controlling Output

19.1	Optimizing Space in Display Lines.	19-1
19.1.1	Adjusting Screen Width and the Columns-Page Setting	19-2
19.1.2	Using the LIST Statement.	19-2
19.1.3	Writing a Simple Procedure to Segment Record Display	19-3
19.1.4	Overriding Column Header Defaults with the PRINT Statement	19-4
19.1.5	Using Edit Strings to Optimize Display Space	19-6
19.1.6	Using Concatenation Characters to Conserve Line Space . . .	19-7
19.2	PRINT Statement Options	19-9
19.2.1	PRINT Statement Format and Print List Elements.	19-9
19.2.2	Using Print List Modifiers	19-13
19.2.3	Sending Output to a File or Printer	19-13

20 Writing Reports

20.1	Entering the REPORT Statement.	20-4
20.2	Controlling Headers and Other Report Settings.	20-5
20.3	Specifying Detail Lines	20-10
20.3.1	Specifying and Formatting Values in a Detail Line.	20-10
20.3.2	Spacing Values in a Detail Line Across the Page	20-10
20.4	Handling Control Groups	20-12
20.4.1	Sorting Records According to Control Group Key Values. . .	20-13
20.4.2	Printing Control Group Headers	20-13
20.4.3	Printing Control Group and Report Summaries	20-13
20.5	Including a Title Page for the Report.	20-16
20.6	Exiting the Report Writer and Correcting Mistakes	20-19

Part VII Appendixes and Index

A DATATRIEVE Keywords

B Sample Record, Table, and View Definitions

C DATATRIEVE Sort Order

D Edit String Characters

Index

Examples

9-1	Sample DATATRIEVE Record Definition	9-2
10-1	Defining a Sample Domain	10-2
11-1	Defining a Data File.	11-1
11-2	Restructuring a Domain to Change File Organization	11-8
11-3	Restructuring a Domain to Change the Record Definition	11-10
12-1	Defining a Dictionary Table	12-2
12-2	Defining a Domain Table.	12-4
13-1	Starting and Ending Access to Data	13-2
14-1	Creating and Using a Collection	14-1
15-1	Including RSEs in Statements	15-2
15-2	Using RSEs in Compound Statements	15-4
15-3	Defining and Using a View.	15-14
16-1	Storing Records Interactively.	16-1
16-2	Storing Records in a Procedure	16-3
16-3	Erasing Records by First Creating a Collection	16-5
16-4	Erasing Records Using a FOR Statement RSE	16-8
16-5	Modifying Records by First Creating a Collection	16-11
16-6	Modifying Records in a FOR Statement RSE	16-13
17-1	Creating a DATATRIEVE Procedure	17-2
18-1	Using Statistical Value Expressions.	18-18
18-2	Using the SUM Statement and Statistical Value Expressions with the CURRENT Collection	18-20
18-3	Storing and Displaying Values in Fields Defined as USAGE DATE .	18-24
18-4	Comparing and Searching for Date Values	18-26
20-1	Sample Report.	20-2
20-2	Using SET Statements to Vary Report Format	20-7
20-3	Varying the Format of Detail Lines	20-11
20-4	Including Control Groups in a Report	20-14
20-5	Creating a Title Page	20-17

Figures

1-1	Organization of PHONES Domain	1-17
2-1	Sample VMS Directory Structure	2-19
4-1	Sample Syntax Diagram	4-2
7-1	CDD Structure	7-2
7-2	Sample CDD	7-3
8-1	Domains and Tables in Sample Personnel System	8-9
9-1	Logical Model of EMPLOYEES_REC	9-3

Tables

2-1	Default File Types	2-11
2-2	Examples of the DELETE Command.	2-13
2-3	System Default Logical Names	2-25
4-1	Notation Used in Syntax Diagrams	4-3
7-1	Specifying Version Numbers	7-6
7-2	SHOW Command Options	7-10
7-3	Access Control Privileges	7-17
7-4	Access Privilege Requirements for Commands and Statements	7-19
8-1	Fields for Personnel System	8-5
9-1	Picture String Characters.	9-10
9-2	Relating Numeric Picture Strings to Stored Values	9-11
9-3	USAGE Clause Options.	9-13
9-4	Editing Text Fields.	9-21
9-5	Editing Numeric Fields	9-21
9-6	Editing Date Fields.	9-22
13-1	Access Options	13-4
13-2	Access Modes	13-4
13-3	Multiuser Access	13-5
18-1	Value Expressions	18-2
18-2	Arithmetic Operators	18-7
18-3	Relational Operators	18-11
18-4	Statistical Functions and Results	18-17
19-1	Print List Elements.	19-12
19-2	Print Item Modifiers	19-13
20-1	Report Writer SET Statement Options	20-6
A-1	DATATRIEVE Keywords	A-1
A-2	DATATRIEVE Function Names	A-5
C-1	DATATRIEVE Sort Order.	C-1
D-1	Edit String Characters	D-1

How to Use This Manual

This manual describes how to use the statements and commands of the VAX DATATRIEVE query language (sometimes simply called DATATRIEVE) to carry out data processing tasks. It provides some tutorial information about describing data and creating procedures for users who are developing data processing skills.

Intended Audience

This manual is intended for users who are in one of two categories:

- Those who have limited or no experience with computer languages and are unfamiliar with the basic elements of DATATRIEVE
- Those who have experience using languages like COBOL or BASIC and want introductory information about DATATRIEVE

Operating System Information

Information about the versions of the operating system and related software that are compatible with this version of VAX DATATRIEVE is included in the VAX DATATRIEVE media kit, in either the Installation Guide or the Before You Install letter.

Contact your DIGITAL representative if you have questions about the compatibility of other software products with this version of VAX DATATRIEVE. You can request the most recent copy of the *VAX System Software Order Table/Optional Software Cross Reference Table*, SPD 28.98.xx, which will verify which versions of your operating system are compatible with this version of VAX DATATRIEVE.

Structure

This manual is divided into seven parts:

- Part I** **Getting Started with DATATRIEVE**
- Provides an overview of concepts important to people just starting to use DATATRIEVE on VMS systems (Chapters 1 to 4).
- Part II** **Controlling Your DATATRIEVE Environment**
- Contains information about startup options, online assistance, and the VAX Common Data Dictionary (Chapters 5 to 7).
- Part III** **Setting Up a Database**
- Leads you through the process of deciding how to organize a database and create the needed data definitions and files (Chapters 8 to 12).
- Part IV** **Data Retrieval and Maintenance**
- Explains the data access options DATATRIEVE offers you and shows you how to store, erase, and modify records (Chapters 13 to 16).
- Part V** **Programming with DATATRIEVE**
- Tells you how to create complex statements and procedures and how to calculate values with DATATRIEVE (Chapters 17 and 18).
- Part VI** **Formatting Displays and Writing Reports**
- Shows you how to improve screen displays and use the DATATRIEVE Report Writer (Chapters 19 and 20).
- Part VII** **Appendixes and Index**
- Contains reference tables for DATATRIEVE keywords, sort order, and editing characters; sample record, table and view definitions; and an index.

Related Documents

For further information on the topics covered in this manual, you can refer to:

- *VAX DATATRIEVE Release Notes*

Includes specific information about the current DATATRIEVE release and contains material added too late for publication in the other DATATRIEVE documentation.

- *VAX DATATRIEVE Installation Guide*

Describes the installation procedure for VAX DATATRIEVE. The manual also explains how to run User Environment Test Packages (UETPs), which test DATATRIEVE product interfaces, such as the interface between DATATRIEVE and Rdb/VMS.

- *VAX DATATRIEVE Guide to Using Graphics*

Introduces the use of DATATRIEVE graphics and contains the reference material for creating DATATRIEVE plots.

- *VAX DATATRIEVE Guide to Writing Reports*

Explains how to use the DATATRIEVE Report Writer.

- *VAX DATATRIEVE User's Guide*

Describes how to use DATATRIEVE interactively. The manual includes information on using DATATRIEVE to manipulate data and on using DATATRIEVE with forms, relational databases, and database management systems. It also describes how to improve performance and how to work with remote data.

- *VAX DATATRIEVE Reference Manual*

Contains reference information for DATATRIEVE.

- *VAX DATATRIEVE Pocket Guide*

Contains quick-reference information about using DATATRIEVE.

- *VAX DATATRIEVE Guide to Programming and Customizing*

Explains how to use the DATATRIEVE Call Interface. The manual also describes how to create user-defined keywords and user-defined functions to customize DATATRIEVE and how to customize DATATRIEVE help and message text.

Conventions

This section explains the conventions for the syntax and symbols used in this manual:

WORD	An uppercase word in a syntax format is a keyword. You must include it in the statement if the clause is used.
word	A lowercase word in a syntax format indicates a syntax element that you supply.
[]	Square brackets enclose optional clauses from which you can choose one or none.
{ }	Braces enclose clauses from which you must choose one alternative.
<code>RET</code>	This symbol indicates the RETURN key. Unless otherwise indicated, end all user input lines in examples by pressing the RETURN key.
<code>TAB</code>	This symbol indicates the TAB key.
<code>CTRLx</code>	This symbol tells you to press the CTRL (control) key and hold it down while pressing a letter key. If you press CTRL/Z, the word Exit appears in reverse video; if you press CTRL/Y, the word Interrupt appears in reverse video. Examples of video output in this book do not include either word; instead the conventions ^Z and ^Y are used.
<code>GOLD-x</code>	This symbol indicates that you press the GOLD key and then a specified letter key consecutively.
" "	These are double quotation marks.
' '	These are single quotation marks.
...	A horizontal ellipsis in syntax formats means you can repeat the previous item. A horizontal ellipsis in examples means that information not directly related to the example has been omitted.

A vertical ellipsis in syntax formats means you can repeat the syntax element from the preceding line.

A vertical ellipsis in examples means that information not directly related to the example has been omitted.

Color

Color in examples shows user input.

Since CDD Version 3.1, CDD path names include a leading underscore. For example:

```
DTR> SHOW DICTIONARY  
The default dictionary is _CDD$TOP.DTR32.WEAGER
```

Examples of the output in DATATRIEVE manuals do not reflect this change. You do not need to enter CDD path names with the leading underscore.

References to Products

VAX DATATRIEVE is a member of the VAX Information Architecture, a group of products that work with each other and with VAX languages conforming to the VAX calling standard to provide flexible solutions for information management problems.

VAX Information Architecture documentation explaining how these products interrelate is included with the VAX CDD documentation. VAX Information Architecture documentation is also available separately. Contact your DIGITAL representative.

The VAX DATATRIEVE documentation to which this manual belongs often refers to products that are part of the VAX Information Architecture by their abbreviated names:

- VAX CDD software is referred to as CDD.
- VAX DATATRIEVE software is referred to as DATATRIEVE.
- VAX DBMS software is referred to as DBMS.
- VAX Rdb/ELN software is referred to as Rdb/ELN.
- VAX Rdb/VMS software is referred to as Rdb/VMS.

- VAX TDMS software is referred to as TDMS.
- VIDA software is referred to as VIDA.

This manual uses the terms relational database or relational source to refer to all three of these products:

- VAX Rdb/ELN
- VAX Rdb/VMS
- VIDA

Technical Changes and New Features

This section describes the technical changes and new features for VAX DATATRIEVE that are documented in this manual.

Version 4.1

Chapters 1 and 3 of this manual include the following new information:

- Using the arrow keys or CTRL/B for line recall
- Using LSE and VAXTPU with DATATRIEVE

Line recall ability is new with Version 4.1. LSE and VAXTPU were new DATATRIEVE features with Version 4.0 and 3.4, respectively.

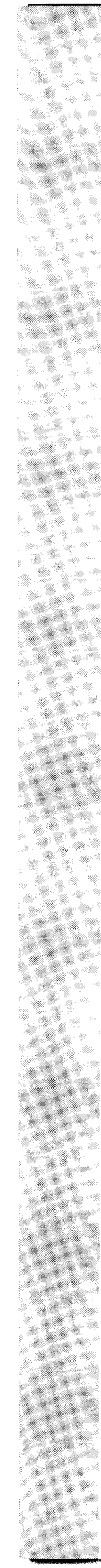
Each new feature was documented in the *VAX DATATRIEVE Release Notes* for its particular release and in online help. With Version 4.1, these features are now included this manual.

All new features since Version 3.0 are described in online help. To read a description of these new features from within DATATRIEVE, refer to the online help:

```
DTR> HELP New_Features
```



Part I
Getting Started with DATATRIEVE



Getting Started with VAX DATATRIEVE 1

This chapter introduces you to DATATRIEVE and the VMS operating system by asking you to enter some commands and statements to perform simple tasks.

If you have limited programming experience, working through the examples in this chapter gives you experience describing, storing, and displaying data with DATATRIEVE. This experience provides a foundation for the more detailed discussion of the same topics in other chapters.

If you are an experienced programmer but have not used a VAX computer system or DATATRIEVE before, simply reading this chapter can help you spot differences between using DATATRIEVE and using other programming languages. See the section later in this chapter for a summary of how DATATRIEVE differs from other computer languages and what advantages it offers.

1.1 What Is DATATRIEVE?

DATATRIEVE is an interactive language and report-writing tool that can help you keep information organized and up-to-date efficiently, quickly, and accurately. You tell DATATRIEVE what to do by typing commands and statements at your keyboard, and DATATRIEVE does the tasks you request. The commands and statements you use are similar to the sentences you would use when asking a person to do the same data management jobs. DATATRIEVE, however, can do these jobs faster, more efficiently, and more accurately than you or any other human being can.

DATATRIEVE is a tool for managing information organized as collections of interrelated data, or databases. You use DATATRIEVE to query and report on a database. DATATRIEVE can access three types of databases:

- File-structured databases that you set up with DATATRIEVE, RMS, or a programming language
- Databases that you create using VAX Rdb/VMS, VAX Rdb/ELN, or VIDA
- Databases that you create using VAX DBMS

Examples in this book show you how to create your own file-structured databases. The *VAX DATATRIEVE User's Guide* explains how to access data stored in DBMS and relational databases. For the remainder of this book, the term *database* is used to refer to data stored in files. This term is sometimes used in other documentation to refer only to data stored by database management systems such as DBMS or the VAX relational database products.

1.2 Create and Use a VMS Directory

This section shows you how to use several system commands important to the DATATRIEVE user. You need no experience with the VMS operating system or with DATATRIEVE. You should have an account on your system and know how to log in and out.

When you log in, you see a dollar sign (\$) prompt on your terminal screen. This prompt indicates that you are at the system command level. The dollar sign (\$) prompt is the VMS default. If you or your system manager change the default with the DCL SET PROMPT command, you see the changed prompt instead. At the system command level, you communicate with the VMS operating system using the DIGITAL Command Language (DCL). Later, when you use DATATRIEVE, you are at DATATRIEVE command level, indicated by a DTR> prompt. At this level, you use the DATATRIEVE language to perform most of your data management tasks. There are a few concepts, however, that you need to know about the system command level before you start working with DATATRIEVE.

When you log in to your system, you are assigned to a VMS directory. A directory is a way of defining and protecting your storage space on the computer system. The data that you will store, modify, and display with DATATRIEVE is stored in files that reside in a VMS directory.

The directory you use when you log in is your *main* directory. It is sometimes referred to as your *login* directory. You can check which VMS directory you are using by typing SHOW DEFAULT at the dollar sign (\$) prompt and then pressing the RETURN key. You can check what your directory contains by typing DIRECTORY and then pressing the RETURN key.

Note

Pressing RETURN tells the system that you want it to process the line you have typed. This book does not use a symbol to tell you when to press RETURN. Therefore, unless the examples in this book indicate otherwise, press the RETURN key to enter each line you type.

The following example shows the results of a DIRECTORY command entered from his main directory by someone with the user name BELL. When you enter the command, you see your user name in place of BELL. In addition, your display might be different from the one in the example. For example, if there is nothing in your directory, the system displays the message “No files found.”

```
$ DIRECTORY
```

```
Directory DBA1:[BELL]
```

```
HELLO.LIS;2          HELLO.LIS;1          LOGIN.COM;1          MAIL.MAI;1
```

```
Total of 4 files.
```

```
$
```

The names of files in a directory listing contain three parts in the following format:

filename.type;version

Filename is the given name of the file, generally what you think of first when you are choosing a name for a file. HELLO, LOGIN, and MAIL are the file names in Bell’s directory display.

Type indicates the kind of file it is. The file types in Bell’s display are .LIS, .COM, and .MAI. Although you can choose up to 39 characters for the type, naming conventions help you identify the type of file you are using. For example, the file type:

- .LIS indicates a file you can print or display
- .COM indicates a procedure you can execute
- .MAI indicates a file that stores messages you receive through the VMS MAIL utility

Version is a decimal number from 1 to 32767 that indicates versions of a file. Whenever you change the contents of a file by editing it, you do not destroy the original contents of the file. You create a file with the same name and type, but a higher version number. Older versions of the file remain in your directory until you decide to get rid of them. (Chapter 2 tells you how to delete files.) In Bell's directory display, all the files except HELLO.LIS have one version. HELLO.LIS has two versions, HELLO.LIS;2 and HELLO.LIS;1.

You can always abbreviate DCL commands when you enter them. This saves typing time and reduces the chance of typographical errors. Shortening commands to the first four letters works for all DCL commands. You can shorten some commands even more, to the first three or even two letters, as long as the abbreviation is not one that could refer to more than one DCL command. The system understands when you type DIR, for example, that you mean DIRECTORY.

Even though abbreviating commands reduces the chances for typographical errors, you will probably make a few during this session. When you make these mistakes at the system command level, you usually receive a message that says a parameter or delimiter is invalid. If you get such a message during this practice session, check what you have typed against the example. You probably spelled a word incorrectly, ran two words together, omitted or changed punctuation, or typed a space where the example did not have one. Simply reenter the command using corrected spelling and format when this happens.

If you are using one of the the VT100-series terminal, another common error is to press the NO SCROLL key in place of the SHIFT key. NO SCROLL "freezes" output to the terminal so that it appears your system has stopped working. Simply press NO SCROLL a second time to clear up this problem. (If you are like most people, you then have to delete all the repeat characters you entered, or, if you pressed RETURN, you will get a few error messages from the system. Simply try the command again when this happens.)

Storing all your computer files in your main directory is analogous to putting all your paper files in the top drawer of a file cabinet. After you accumulate more than a few files, it is difficult to find something when you want to use it. A well-organized file cabinet keeps unrelated files in separate drawers. Similarly, you can append subdirectories to your main VMS directory to store files related to a specific task or subject matter.

The data files and other files that result from this session are temporary. It is best, therefore, to create a VMS subdirectory to keep them apart from the other files you are using. Use the CREATE/DIRECTORY command to do this. The following example creates the subdirectory PRACTICE for user BELL. When you enter the CREATE/DIRECTORY command at your own terminal, substitute your user name for BELL. You can substitute another name for PRACTICE if you want. If you choose another name, limit it to no more than 39 characters, and, for now at least, use only letters of the alphabet.

```
$ CREATE/DIRECTORY [BELL.PRACTICE]
$
```

If you enter the DIRECTORY command again, you see your subdirectory listed with the file type .DIR:

```
$ DIR

Directory DBA1:[BELL]

HELLO.LIS;2          HELLO.LIS;1          LOGIN.COM;1          MAIL.MAI;1
PRACTICE.DIR;1

Total of 5 files.
$
```

Use the SET DEFAULT command to set your directory location to your subdirectory. Just as you cannot see the contents of a drawer in a file cabinet or file anything there unless you open it first, you cannot see the contents of a directory unless you either set your location to that directory or include the directory name in your commands.

When you refer to a subdirectory, you type the name of its parent, your main directory, then type a period (.), and finally the name you chose for it. In addition, note that you enclose a directory name within square brackets ([]). The following example shows how user BELL sets his directory location to his subdirectory PRACTICE and then checks its contents. In the example, MISC\$DISK is a system logical name that stands for the device DBA1:. Chapter 2 tells you more about logical names.

```
$ SET DEFAULT [BELL.PRACTICE]
$ SH DEF
$ MISC$DISK:[BELL.PRACTICE]
$ DIR
No files found.
$
```

You now have an empty directory in which to store your practice data files.

If you type a directory name incorrectly in a SET DEFAULT command and the mistake is inside the brackets, the system does not return an error message. However, you receive a directory not found message if you try to use the directory. In this case, you must enter another SET DEFAULT command with a correct directory name. If you ever get totally lost within a VMS directory structure, you can always type SET DEFAULT SYS\$LOGIN to get back to the login directory.

1.3 Obtain a CDD Directory and Start DATATRIEVE

The first time you use DATATRIEVE, you should run the NEWUSER program. This program:

- Copies some sample data files into the VMS directory you are currently using
- Defines a Common Data Dictionary (CDD) directory for you
- Copies some sample data definitions into that dictionary directory

Just as you use an assigned VMS directory when you log in to your system, you use an assigned dictionary directory when you run DATATRIEVE. Each sample data file (stored on the system command level in a VMS directory) has corresponding record and domain definitions (stored on the DATATRIEVE command level in a dictionary directory). Just as the VMS directory defines and protects your storage space for files, the dictionary directory defines and protects your storage space for data definitions.

To run the NEWUSER program, make sure you are using the VMS subdirectory you created for this practice session and enter the following command:

```
$ @DTR$LIBRARY:NEWUSER
```

The program responds with the following information:

```
NEWUSER helps new users to get started with DATATRIEVE. It gives
you the necessary files to perform the introductory examples in the
VAX DATATRIEVE Handbook and the examples in the VAX DATATRIEVE
User's Guide and Reference Manual.
```

```
NEWUSER is working... It will take a few minutes.
All data copied successfully.
```

```
The following commands have been defined for you but you will need to
add them to your LOGIN.COM file for the next time you log in:
```

```
$      dtr32 == "$sys$system:DTR32.EXE"
$      assign/process "cdd$top.dtr$users.bell" cdd$default
If you need help, see the person responsible for DATATRIEVE on your system.
```

```
To invoke DATATRIEVE just type:      DTR32
```

The results that appear for user Bell are not exactly the ones you will get when you run the NEWUSER program. If the display indicates that NEWUSER has run successfully, write down the line from your display that begins with dtr32 == and the one that begins with assign/process. Add these lines to the LOGIN.COM file in your main directory.

If no one defined a LOGIN.COM file for you when setting up your system account or if you do not yet know how to edit a file, you can use the entries from the following example. This example uses the EDT editor. If your LOGIN.COM file already exists, you will see the first line of that file in place of "Input file does not exist":

```
$ SET DEFAULT SYS$LOGIN
$ EDIT LOGIN.COM
Input file does not exist
[EOB]
*C
```

At this point you are using an editing *buffer*. You can think of a buffer as an unnamed section of storage space created for you to do some work. The [EOB] marker indicates end of buffer. As you add text to the buffer, this marker moves to a position just below your last line of input.

You can position the cursor using the arrow keys on your keyboard. The cursor indicates where the characters you enter will go when you start typing. If you position the cursor at the end of a line and press the RETURN key, you can open up space below to enter a new input line. If you position the cursor at the beginning of a line and press the RETURN key, you can open up space above to enter a new input line. To correct typographical errors, you can press the DELETE key to erase characters (including an accidental RETURN.) Pressing the DELETE key erases the character just before the cursor. There are more efficient ways to use an editor, but these suggestions will do the job until you learn more about editing.

When you finish including the two lines in the file, press CTRL/Z (hold down the CTRL key and Z at the same time). At the asterisk (*) prompt, type EXIT and press RETURN. When you enter the EXIT command, the EDT editor creates the LOGIN.COM file or, if it already exists, a new version of that file.

Later you can refer to the chapter on using editors with DATATRIEVE to learn how to use an editor to create and modify files. You can also read the chapter on VMS concepts for an explanation of what command files such as LOGIN.COM can do for you.

If the display indicates that the NEWUSER program “aborted” or prints messages telling you that definitions were not copied, the person who installed DATATRIEVE on your system selected a different default dictionary setting than the one the NEWUSER program uses (CDD\$TOP.DTR\$USERS). In this case, you will have to ask the person responsible for the CDD software on your system to set up a dictionary directory for you. After this is done, you can enter the following commands, substituting your assigned dictionary directory name for CDD\$TOP.DTR\$USERS.BELL:

```
$ ASSIGN "CDD$TOP.DTR$USERS.BELL" CDD$DEFAULT
$ @DTR$LIBRARY:NEWUSER
```

When you insert in your LOGIN.COM file the lines NEWUSER tells you to add, the assignment to CDD\$DEFAULT takes effect whenever you log in to your system.

Set your default VMS directory to the one you created for this practice session. You can now run DATATRIEVE with the following command:

```
$ DTR32
VAX Datatrieve V4.1
DEC Query and Report System
Type HELP for help
DTR>
```

The startup display indicates that you are in DATATRIEVE and tells you what version of the product you are running. The DTR> prompt tells you that you are at DATATRIEVE command level and that DATATRIEVE is ready for you to enter instructions.

Note

You can use the arrow keys or CTRL/B within to recall what you have typed on previous lines. See the chapter on using editors with DATATRIEVE for more information.

Use the SHOW DICTIONARY command to check your dictionary location. (The DATATRIEVE SHOW command is analogous to the SHOW and DIRECTORY commands you use at system command level.) The following example shows the result of the SHOW DICTIONARY command when entered by user Bell:

```
DTR> SHOW DICTIONARY
The default directory is 'CDD$TOP.DTR$USERS.BELL
DTR>
```

You can see that when you invoke DATATRIEVE, your dictionary location is the directory you assigned to CDD\$DEFAULT in your LOGIN.COM file.

Use the SHOW DOMAINS, RECORDS command to display the data definitions that the NEWUSER program copied into your dictionary directory:

```
DTR> SHOW DOMAINS, RECORDS
Domains:
    FAMILIES;1      OWNERS;1          PERSONNEL;1      PETS;1
    PROJECTS;1     YACHTS;1
Records:
    FAMILY_REC;1   OWNER_RECORD;1   PERSONNEL_REC;1  PET_REC;1
    PROJECT_REC;1 YACHT;1
DTR>
```

As you can see, the definitions in a dictionary directory have a version number, just like the files stored in a VMS directory. An earlier version of a definition serves the same purpose as an earlier version of a file—it gives you something to fall back on if you cannot use any changes you make to the definition.

1.4 Look at Some Sample Definitions and Data

As mentioned earlier, the NEWUSER program copies sample data definitions into your default dictionary directory. For each data file NEWUSER copied into your VMS directory, NEWUSER copied into your default dictionary directory two data definitions: a **record definition** and a **domain definition**.

- A record definition tells you how each record in a data file is divided into more elementary parts (fields) and, among other things, how much space each part requires in storage. A record definition is a template that both you and DATATRIEVE use to identify the fields in a record. It is analogous to a blank job application form before a prospective employee fills in the blanks.
- A domain definition relates a record definition to a data file and specifies a name for this relationship. You use the domain name as the source or destination for data when you enter DATATRIEVE commands and statements. The domain name tells DATATRIEVE that the specified record definition is the way to interpret the records in the corresponding data file.

You can look at these definitions with the SHOW command.

In the following example, the command SHOW PERSONNEL reveals the contents of the domain definition PERSONNEL. Note that it specifies the name of a data file, PERSON.DAT, that the NEWUSER program copied into your VMS subdirectory. Because NEWUSER puts only a file name and type in the domain definition, DATATRIEVE expects to find the data file in the VMS directory from which you invoked DATATRIEVE. Chapter 2 discusses the more complete file specification you can include in domain definitions to get around this default. The domain definition also specifies a record definition, PERSONNEL_REC, that describes each record in that file:

```
DTR> SHOW PERSONNEL
DOMAIN PERSONNEL USING PERSONNEL_REC ON PERSON.DAT;

DTR>
```

The command SHOW PERSONNEL_REC displays the contents of the record definition:

```
DTR> SHOW PERSONNEL_REC
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
        DEFAULT VALUE IS "TODAY".
    05 SALARY PIC IS 9(5)
        EDIT_STRING IS $$$,$$$ .
    05 SUP_ID PIC IS 9(5)
        MISSING VALUE IS 0.
;

DTR>
```

If you are not a programmer, you probably do not understand all parts of a record definition. These are described for you in later chapters. For now, note that a record definition specifies:

- The fields in each record (such as `EMPLOYEE_NAME` and `SALARY`)
- The order in which the fields are stored in the record (`ID`, then `EMPLOYEE_STATUS`, and so forth)
- The type of characters that each field can contain (`PIC X` when it can contain most keyboard characters and `PIC 9` when it can contain only numbers)
- The number of characters each field can contain (`PIC 9(5)` and `PIC X(11)`, for example)

If you wish, you can look at some of the other definitions supplied by the `NEWUSER` program. `SHOW DOMAINS` gives you a listing of the domains, `SHOW name-of-domain` displays the data file and corresponding record definition, and `SHOW name-of-record` gives you some information about the data in the file.

To access data, you must type `READY`, followed by the domain name that applies to the file where the data is stored. The following command tells `DATATRIEVE` to ready the `PERSONNEL` domain for your use:

```
DTR> READY PERSONNEL
DTR>
```

After you ready a domain, you can display records with the `PRINT` command. To stop the display temporarily, press the `NO SCROLL` key if you are using a VT100-type terminal, the `HOLD SCREEN` key if you are using a VT200-type terminal, or `CTRL/S` if you are using another type of terminal. To start the display again after a temporary stop, press `NO SCROLL` a second time (for VT100- or VT200-type terminals) or `CTRL/Q` (for the others).

You can enter CTRL/C to stop a display altogether if you do not want to see all the records:

```
DTR> PRINT PERSONNEL
```

ID	STATUS	FIRST NAME	LAST NAME	DEPT	START DATE	SALARY	SUP ID
00012	EXPERIENCED	CHARLOTTE	SPIVA	TOP	12-Sep-1972	\$75,892	00012
00891	EXPERIENCED	FRED	HOWL	F11	9-Apr-1976	\$59,594	00012
02943	EXPERIENCED	CASS	TERRY	D98	2-Jan-1980	\$29,908	39485
12643	TRAINEE	JEFF	TASHKENT	C82	4-Apr-1981	\$32,918	87465
32432	TRAINEE	THOMAS	SCHWEIK	F11	7-Nov-1981	\$26,723	00891
34456	TRAINEE	HANK	MORRISON	T32	1-Mar-1982	\$30,000	87289
38462	EXPERIENCED	BILL	SWAY	T32	5-May- <u>CTRL/C</u>		

```
^C
```

Execution terminated by operator.

The following example illustrates a few of the commands and options you have when displaying records.

The SHOW FIELDS command displays a list of fields in readied domains. SHOW FIELDS displays not only the full field names, but also any query names (in parentheses) that have been defined for the fields. Using query names in your DATATRIEVE statements can save you some keystrokes.

Note that if you press RETURN at a point in your statement where DATATRIEVE knows the statement is incomplete, you receive a message telling you what DATATRIEVE expects next. The CON> prompt indicates that DATATRIEVE is waiting for you to continue:

```
DTR> SHOW FIELDS FOR PERSONNEL
```

```
PERSONNEL
PERSON
  ID          (Number, primary key)
EMPLOYEE_STATUS (STATUS) (Character string)
EMPLOYEE_NAME (NAME)
  FIRST_NAME (F_NAME) (Character string)
  LAST_NAME (L_NAME) (Character string)
DEPT          (Character string)
START_DATE   (Date)
SALARY       (Number)
SUP_ID       (Number)
```

DTR> PRINT NAME, DEPT, STATUS OF PERSONNEL

FIRST NAME	LAST NAME	DEPT	STATUS
CHARLOTTE	SPIVA	TOP	EXPERIENCED
FRED	HOWL	F11	EXPERIENCED
CASS	TERRY	D98	EXPERIENCED
JEFF	TASHKENT	C82	TRAINEE
THOMAS	SCHWEIK	F11	TRAINEE
HANK	MORRISON	T32	TRAINEE
BILL	SWAY	T32	EXPERIENCED
JOANNE	FREIBURG	E46	EXPERIENCED
DEE	TERRICK	D98	EXPERIENCED
GAIL			

CTRL/C

^C

Execution terminated by operator.

DTR> PRINT PERSONNEL WITH DEPT = "T32"

ID	STATUS	FIRST NAME	LAST NAME	DEPT	START DATE	SALARY	SUP ID
34456	TRAINEE	HANK	MORRISON	T32	1-Mar-1982	\$30,000	87289
38462	EXPERIENCED	BILL	SWAY	T32	5-May-1980	\$54,000	00012
48573	TRAINEE	SY	KELLER	T32	2-Aug-1981	\$31,546	87289
83764	EXPERIENCED	JIM	MEADER	T32	4-Apr-1980	\$41,029	87289

DTR> PRINT DEPT, NAME, STATUS OF

[Looking for name of domain, collection, or list]

CON> PERSONNEL SORTED BY DEPT

DEPT	FIRST NAME	LAST NAME	STATUS
C82	JEFF	TASHKENT	TRAINEE
C82	ANTHONY	IACOBONE	EXPERIENCED
C82	DAN	ROBERTS	EXPERIENCED
C82	BRUNO	DONCHIKOV	EXPERIENCED
C82	RANDY	PODERESIAN	EXPERIENCED
D98	CASS	TERRY	EXPERIENCED
D98	BART	HAMMER	TRAINEE
D98	MARY	NALEVO	EXPERIENCED
D98	DEE	TERRICK	EXPERIENCED
E46	GAIL	CASSIDY	EXPERIENCED
E46	JOANNE	FREIBURG	EXPE

CTRL/C

^C

Execution terminated by operator.

DTR>

At this point, you might want to try readying and displaying records from some of the sample domains other than PERSONNEL. If you receive a message that a name is undefined or used out of context, it probably means that you made one of the following mistakes:

- You tried to print records without first readying the domain. (If you forget which domains you have readied, you can enter SHOW READY to find out.)
- You typed a domain or field name incorrectly.
- You used a field name that is not found in the specified domain.

To correct your mistake, you can retype the entire statement or command with corrections. DATATRIEVE, however, allows you to edit the last statement or command in either of two ways:

- You can type EDIT and press the RETURN key. This invokes EDT, the default DATATRIEVE editor. If you are familiar with EDT, you can then use it to modify your input, even adding new commands and statements. When you exit the editor, DATATRIEVE processes what you typed in the edit buffer.
- You can use the arrow keys or CTRL/B to recall previous lines. You can then correct any errors you made and press the RETURN to execute the line. Line recall is particularly useful if you are not familiar with an editor.

See the chapter in this book on using editors for more information.

1.5 End Data Access and Exit DATATRIEVE

When you finish working with a domain, you can use the FINISH command to end access to its data:

```
DTR> READY PERSONNEL
DTR> SHOW READY
Ready sources:
  PERSONNEL: Domain, RMS indexed, protected read
              <CDD$TOP.DTR$USERS.BELL.PERSONNEL;1>
No loaded tables.

DTR> FINISH PERSONNEL
DTR> SHOW READY
No ready sources.
No loaded tables.

DTR> PRINT PERSONNEL
"PERSONNEL" is undefined or used out of context.
DTR>
```

Finishing domains you are no longer using frees the system resources needed to keep the data file available to you.

When you want to end your DATATRIEVE session, you can simply type EXIT or press CTRL/Z at the DTR> prompt. When you exit, DATATRIEVE finishes any readied domains for you and returns you to system command level. If you plan to continue with the examples in the next section, do not exit DATATRIEVE at this point.

If you do try the EXIT command, however, you can run DATATRIEVE again by entering DTR32 at the dollar sign (\$) prompt. If you log out or use the SET DEFAULT command, be sure to set your default to the VMS directory you created for practice before running DATATRIEVE again:

```
$ SET DEF [BELL.PRACTICE]
$ DTR32
```

1.6 Create a Dictionary Subdirectory

The dictionary directory structure at DATATRIEVE command level is analogous to the VMS directory structure at system command level. Unlike the VMS directory structure, however, where each login directory is a top-level directory, the dictionary directory you are assigned as a DATATRIEVE user is not a top-level directory. CDD\$TOP is the name of the top-level directory in the Common Data Dictionary. All other directories can trace their parentage back to that directory. (Thinking of the structure as a family tree, with CDD\$TOP as the beginning, might be helpful in remembering how things relate to one another.)

For this reason, names of directories and definitions in the dictionary are called **path names**. The full path name of your dictionary directory starts with CDD\$TOP and includes the names of all the directories that lead back to CDD\$TOP. Each name is separated from another by a period. For example, CDD\$TOP.DTR\$USERS.BELL is the full path name for the directory assigned to user Bell. You can see the full path name of the directory you are using by entering the SHOW DICTIONARY command.

The full name of each data definition you store in a dictionary directory also starts with CDD\$TOP and includes the names of all the directories that lead to its location. The full name of the PERSONNEL domain definition copied into Bell's assigned directory by the NEWUSER program is CDD\$TOP.DTR\$USERS.BELL.PERSONNEL.

Chapter 7 explains more about CDD structure and tells you when you need to specify full path names and how you can abbreviate them. The examples in this chapter rarely contain full path names because DATATRIEVE can find what you want to use in the dictionary directory at which you are currently located.

You can append subdirectories to the CDD directory created for you by the NEWUSERS program or assigned to you by the CDD manager on your system. This allows you to organize your definition storage area so that things are easy to find. This may not seem important now, but will be when your directory listing grows larger.

You create a dictionary subdirectory by using the DEFINE DICTIONARY command. The following example shows how user Bell appends a subdirectory to the one he uses when he first invokes DATATRIEVE:

```
DTR> SHOW DICTIONARY
The default directory is CDD$TOP.DTR$USERS.BELL
```

```
DTR> DEFINE DICTIONARY PRACTICE
DTR>
```

Use the SET DICTIONARY command to set your dictionary location to your new subdirectory. At the new location, using SHOW DICTIONARY reveals the full path name of your subdirectory:

```
DTR> SET DICTIONARY PRACTICE
DTR> SHOW DICTIONARY
The default directory is CDD$TOP.DTR$USERS.BELL.PRACTICE
```

```
DTR>
```

1.7 Create a Simple Application

Earlier in this chapter, you saw how to access data when the data file and domain and record definitions were already in place. This section shows you how to create your own data file and definitions. You will:

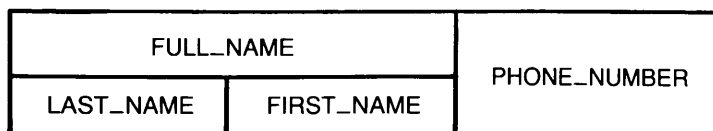
- Create the definitions and data file needed to store a list of phone numbers
- Store a few records in your newly created domain
- Print the results
- Modify some records
- Add a field to the records
- Create a DATATRIEVE table
- Create a DATATRIEVE procedure

1.7.1 Use ADT to Create Data Definitions and a File

The DATATRIEVE Application Design Tool (ADT) provides a fast way to create a database for users who have never created a record definition or data file before, and who do not yet know how to execute the steps needed to create a database.

Experienced programmers can use ADT to save time when creating record and domain definitions because ADT automatically includes much of what is needed in those definitions. After the definitions exist, programmers can edit them to add any DATATRIEVE options that ADT does not provide but that they want to include.

In this section, you create a small database with the domain name PHONES to contain a list of names and phone numbers. Figure 1-1 illustrates the organization of this domain.



MK-01570-00

Figure 1-1: Organization of PHONES Domain

Figure 1-1 shows four field names: FULL_NAME, LAST_NAME, FIRST_NAME, and PHONE_NUMBER. Of these, FULL_NAME is a **group field** and LAST_NAME, FIRST_NAME, and PHONE_NUMBER are **elementary fields**.

As you might already know, group fields contain fields that are related to one another. These fields can be elementary fields or other group fields. Generally, you decide to create a group field when you know that you want to retrieve two or more fields at a time by using one name, but also want to use each of the subordinate fields separately. FULL_NAME is a common example of this situation.

Elementary fields do not contain other fields. Later, you will see that ADT prompts you to supply more information about elementary fields than it does for group fields. That is because the elementary fields generate the rules that apply to data storage—how many and what kind of characters you are allowed to store in a given section of a record.

You run ADT by typing ADT at the DTR> prompt and pressing RETURN. The following example shows the ADT session that creates the PHONES domain. The example does not show all that is displayed on your screen as you create the domain; however, it does contain all the questions and the responses you should make:

DTR> ADT

ADT helps you set up a structure for your data. After a series of questions and responses, ADT constructs the definitions for a DATATRIEVE domain, record, and data file. You can have ADT write these definitions into your dictionary.

Do you want detailed prompts? (YES or NO) : YES

Enter the name for your domain.
Start with a letter and use letters,
digits, hyphens (-), or underscores (_).
(No spaces or tabs) : PHONES

The data for the domain PHONES will be stored in a file
with the name you specify.

Enter the name for PHONES's data file : PHONES.DAT

Enter the first field name in record for PHONES.
Start with a letter and use only letters, digits,
hyphens (-), or underscores (_).
(No spaces or tabs.) : FULL_NAME

Enter an abbreviation you can use
as a query name for FULL_NAME : NAME

What's in FULL_NAME -- DATE
PERCENT
MONEY
NUMBERS used in arithmetic
CHARACTERS
GROUP?

Enter one of these words or its first letter : G

Enter the first field name in record for FULL_NAME.
Start with a letter and use only letters, digits,
hyphens (-), or underscores (_).
(No spaces or tabs.) : LAST_NAME

Enter an abbreviation you can use
as a query name for LAST_NAME : L

What's in LAST_NAME -- DATE
PERCENT
MONEY
NUMBERS used in arithmetic
CHARACTERS
GROUP?

Enter one of these words or its first letter : C

Enter the maximum number of characters for LAST_NAME : 20

Do you want to define more fields in FULL_NAME ? (YES or NO) : YES

Enter the name of the next field in FULL_NAME : FIRST_NAME

Enter an abbreviation you can use
as a query name for FIRST_NAME : F

What's in FIRST_NAME -- DATE
PERCENT
MONEY
NUMBERS used in arithmetic
CHARACTERS
GROUP?

Enter one of these words or its first letter : C

Enter the maximum number of characters for FIRST_NAME : 15

Do you want to define more fields in FULL_NAME ? (YES or NO) : NO

Do you want to define more fields in PHONES ? (YES or NO) : YES

Enter the name of the next field in PHONES : PHONE_NUMBER

Enter an abbreviation you can use
as a query name for PHONE_NUMBER : NUM

What's in PHONE_NUMBER -- DATE
PERCENT
MONEY
NUMBERS used in arithmetic
CHARACTERS
GROUP?

Enter one of these words or its first letter : C

Enter the maximum number of characters for PHONE_NUMBER : 8

Do you want to define more fields in PHONES ? (YES or NO) : NO

Do you want the data file to be indexed?
You can't modify primary keys in indexed files.
You can't erase records from sequential files. (YES or NO) : YES

Enter the name of the primary key field.
You cannot modify data stored in a primary key field. : L

Do you want the primary key fields in different records
to have duplicate values? (YES or NO) : YES

Do you want to specify an alternate index key? (YES or NO) : NO

Do you want ADT to add the domain and record definitions
to your current default dictionary directory? (YES or NO) : YES
[Record is 43 bytes long.]

Do you want to define another domain? (YES or NO) : NO

DTR)

If you enter the SHOW DOMAINS, RECORDS command, you now see PHONES and PHONES_REC listed in your dictionary directory. You can enter SHOW PHONES and SHOW PHONES_REC to see what these definitions contain. If you exit DATATRIEVE and enter the DIRECTORY command, you see PHONES.DAT listed in your PRACTICE directory. (If you do exit DATATRIEVE, however, remember to set your dictionary location to your dictionary subdirectory before you continue with the next section.)

1.7.2 Store Records

You must ready the PHONES domain for write access before you can store records:

```
DTR> READY PHONES WRITE
DTR>
```

You can now enter data. The following example stores five records in PHONES. Note that DATATRIEVE prompts you to enter values for each elementary field in each of the five records. Feel free to enter values of your own choosing. Try, for one field, to enter a value that is too large for the field and see what happens:

```
DTR> REPEAT 5 STORE PHONES
Enter LAST_NAME: BELL
Enter FIRST_NAME: LISA
Enter PHONE_NUMBER: 883-8275
Enter LAST_NAME: LINTE
Enter FIRST_NAME: JANE
Enter PHONE_NUMBER: 881-2461
Enter LAST_NAME: CLERC
Enter FIRST_NAME: PHYLLIS
Enter PHONE_NUMBER: 884-9907
Enter LAST_NAME: SCHUTZ
Enter FIRST_NAME: BONNIE
Enter PHONE_NUMBER: 567-8712
Enter LAST_NAME: WINTLOW
Enter FIRST_NAME: JOHN
Enter PHONE_NUMBER: 8890-6789
Truncation during assignment.
Re-enter PHONE_NUMBER: 880-6789
DTR>
```

1.7.3 Display Data

The following example illustrates a few variations you can use when displaying records. Do not worry if you do not understand how some of these options work or what they are called. Chapter 14 discusses the FIND statement. Chapter 19 explains in greater detail how you can adjust display formats:

```
DTR> SHOW FIELDS FOR PHONES
```

```
PHONES
```

```
  PHONES_REC
```

```
    FULL_NAME (NAME)
```

```
      LAST_NAME (L) (Character string, primary key)
```

```
      FIRST_NAME (F) (Character string)
```

```
    PHONE_NUMBER (NUM) (Character string)
```

```
DTR> PRINT PHONES
```

LAST NAME	FIRST NAME	PHONE NUMBER
BELL	LISA	883-8275
CLERC	PHYLLIS	884-9907
LINTE	JANE	881-2461
SCHUTZ	BONNIE	567-8712
WINTLOW	JOHN	880-6789

```
DTR> FIND PHONES
```

```
[5 records found]
```

```
DTR> PRINT ALL FULL, NUM
```

	PHONE NUMBER
LISA BELL	883-8275
PHYLLIS CLERC	884-9907
JANE LINTE	881-2461
BONNIE SCHUTZ	567-8712
JOHN WINTLOW	880-6789

```
DTR> PRINT ALL FLNUM, SKIP
```

```
LISA BELL 883-8275
```

```
PHYLLIS CLERC 884-9907
```

```
JANE LINTE 881-2461
```

```
BONNIE SCHUTZ 567-8712
```

```
JOHN WINTLOW 880-6789
```

```
DTR>
```

1.7.4 Change Field Values

You can easily change the values in a record. The following example specifies NUM as the field to be changed and illustrates only one of the many ways to modify a record:

```
DTR> READY PHONES MODIFY
DTR> FIND PHONES WITH LAST_NAME = "LINTE"
[1 record found]
DTR> SELECT
DTR> PRINT
```

LAST NAME	FIRST NAME	PHONE NUMBER
LINTE	JANE	881-2461

```
DTR> MODIFY NUM
Enter PHONE_NUMBER: 889-3456
DTR> PRINT
```

LAST NAME	FIRST NAME	PHONE NUMBER
LINTE	JANE	889-3456

```
DTR>
```

Now try to change a value for LAST_NAME and see what happens:

```
DTR> FIND PHONES WITH LAST_NAME = "BELL"
[1 record found]
DTR> SELECT
DTR> MODIFY LAST_NAME
Enter LAST_NAME: WARTON
You cannot modify the value of an RMS key field that doesn't allow changes.
DTR> PRINT
```

LAST NAME	FIRST NAME	PHONE NUMBER
BELL	LISA	883-8275

```
DTR>
```

If you remember when you created the PHONES domain and were specifying index keys for the file, ADT told you that you cannot change the value of a primary key. You might have wondered then what you would do if someone's name changed, for example, after marriage. To get around the restriction that you cannot change a primary key value, you can erase the record and store it again with the changed value for LAST_NAME. This is not a real problem for a small record, but it would be if your record were larger and you needed to change that key value frequently. Chapter 11 tells you more about primary keys and how to make the best choices when selecting keys for a file.

1.7.5 Change Domain Structure

Suppose that after you create a domain and store quite a few records in it, you decide you want to rearrange the order of the fields in the record, add a new field to the record, or change the index keys for the data file. In other words, you want to *restructure* your domain.

If you edit the record definition to make changes in record structure or size, it no longer corresponds to the existing data file and DATATRIEVE can no longer use it to retrieve data from the file. Similarly, you cannot change key options for a file without creating a new and empty version of the file. When you are creating your own domains, you can expect to change your mind about record and file organization more than once. DATATRIEVE allows you to make these changes without losing data you have already stored by performing a simple restructure operation. See the chapter on defining data files for more details on this topic.

The following example shows you how to restructure the PHONES domain to add a field, AREA_CODE, to the record definition. The FINISH command is included because you readied PHONES for MODIFY access in the last exercise. EDIT invokes the EDT editor by default, but you can invoke one of several different editors if you choose. See the chapter on using editors with DATATRIEVE for more information.

When you edit the record definition, add the AREA_CODE field between the FIRST_NAME and PHONE_NUMBER fields. Enter the SHOW PHONES_REC command to see the revised definition.

```
DTR> FINISH PHONES
DTR> READY PHONES AS OLD
DTR> EDIT PHONES_REC
```

[Record is 46 bytes long.]

```
DTR> SHOW PHONES_REC
RECORD PHONES_REC USING
01 PHONES_REC.
   05 FULL_NAME      QUERY_NAME IS NAME.
      10 LAST_NAME   PIC X(20)
          QUERY_NAME IS L.
      10 FIRST_NAME  PIC X(15)
          QUERY_NAME IS F.
   05 AREA_CODE     PIC X(3)
          QUERY_NAME IS AREA.
   05 PHONE_NUMBER  PIC X(8)
          QUERY_NAME IS NUM.
```

```
DTR> DEFINE FILE FOR PHONES KEY = LAST_NAME (DUP)
DTR> READY PHONES AS NEW WRITE
DTR> NEW = OLD
DTR> FINISH
DTR> READY PHONES
DTR> PRINT PHONES
```

LAST NAME	FIRST NAME	AREA CODE	PHONE NUMBER
BELL	LISA		883-8275
CLERC	PHYLLIS		884-9907
LINTE	JANE		884-5678
SCHUTZ	BONNIE		567-8712
WINTLOW	JOHN		880-6789

```
DTR>
```

You can see that space has been added to the record so you can store area codes. Do not modify your records yet to add area code values.

1.7.6 Create a Table

If you have some programming experience, you might assume the term “table” refers to repeating fields (as in a BASIC array or a COBOL table). In DATATRIEVE terminology, however, repeating fields are called **lists**. DATATRIEVE **tables** store sets of paired values apart from other data definitions.

In this section, you create a **dictionary table** to relate an area code value with the corresponding town and state. A dictionary table stores value pairs in the table definition itself. DATATRIEVE also lets you create **domain tables**. A domain table stores pairs of field names that let you relate data stored in one domain to data stored in another domain. Chapter 12 explains tables more completely and shows a variety of ways to use them. For now, you will learn one way to use one kind of table.

You create a table with the DEFINE TABLE command. Enter the following lines to store AREA_CODE_TABLE in the dictionary you are currently using. The DFN> prompt tells you that DATATRIEVE expects more input for the definition. The END_TABLE entry tells DATATRIEVE you are finished with the definition:

```
DTR> SET NO PROMPT
DTR> DEFINE TABLE AREA_CODE_TABLE
DFN> QUERY_HEADER IS "STATE"
DFN> "603" : "NH"
DFN> "617" : "MA"
DFN> "201" : "NJ"
DFN> ELSE "OOPS"
DFN> END_TABLE
DTR>
DTR> SHOW TABLES
Tables:
      AREA_CODE_TABLE;1
```

Obviously, this is an incomplete list of area codes, but it is sufficient for demonstration purposes.

You can see how your table works with some simple PRINT statements that include a VIA clause. Note how the value associated with the ELSE clause tells you when an area code value is not listed in the table:

```
DTR> PRINT "603" VIA AREA_CODE_TABLE

STATE

NH

DTR> PRINT "111" VIA AREA_CODE_TABLE

STATE

OOPS
```


Now modify your PHONES records to include some area codes, making sure that you include at least one area code that is not in the table. The last time you modified data in PHONES, you only wanted to change one record. The following example includes a FOR statement (read it as "FOR every PHONES record"), so DATATRIEVE lets you add all the area codes:

```
DTR> READY PHONES MODIFY
DTR> FOR PHONES
CON> BEGIN
CON> PRINT
CON> MODIFY USING AREA_CODE = *.AREA_CODE
CON> PRINT
CON> END
```

LAST NAME	FIRST NAME	AREA CODE	PHONE NUMBER
BELL	LISA		883-8275

Enter AREA_CODE: 603

LAST NAME	FIRST NAME	AREA CODE	PHONE NUMBER
BELL	LISA	603	883-8275
CLERC	PHYLLIS		884-9907

Enter AREA_CODE: 603

CLERC	PHYLLIS	603	884-9907
LINTE	JANE		884-5678

Enter AREA_CODE: 603

LINTE	JANE	603	884-5678
SCHUTZ	BONNIE		567-8712

Enter AREA_CODE: 617

SCHUTZ	BONNIE	617	567-8712
WINTLOW	JOHN		880-6789

Enter AREA_CODE: 205

WINTLOW	JOHN	205	880-6789
---------	------	-----	----------

You can now use a VIA AREA_CODE_TABLE expression to display the state that corresponds to the area code for each record. PHONES_REC, as it appears in the following PRINT statement, refers to the top level field in the record definition rather than the record name as it is stored in the dictionary:

```
DTR> FIND PHONES
[5 records found]
DTR> PRINT ALL PHONES_REC, AREA_CODE VIA AREA_CODE_TABLE
```

LAST NAME	FIRST NAME	AREA CODE	PHONE NUMBER	STATE
BELL	LISA	603	883-8275	NH
CLERC	PHYLLIS	603	884-9907	NH
LINTE	JANE	603	884-5678	NH
SCHUTZ	BONNIE	617	567-8712	00PS
WINTLOW	JOHN	205	880-6789	00PS

You might ask at this point why you would not simply store the data contained in AREA_CODE_TABLE in the PHONES domain and save yourself a lot of keystrokes. It is true that tables make little sense when they store information that applies to only one domain. Tables, however, can save you a great deal of storage redundancy when they contain data that you use with more than one domain. Tables also help you validate fields that must be stored in more than one domain. (In a set of domains used by the personnel department in a company, for example, the employee number would need to be stored in more than one domain.)

1.7.7 Write a Procedure

You create a DATATRIEVE procedure to store a set of commands and statements that you plan to use more than once. In one sense, a procedure is like a program written in a language like BASIC or COBOL because, after you create it, you simply execute it to get a job done.

The following example stores the procedure PHONES_REPORT in your current dictionary directory. Writing reports is not the only job you might want to do with procedures, but it is one of the most common. Do not worry if you do not understand what all the statements in the example do for you. Chapter 20 explains report writing more fully for you. Right now, this gives you some initial practice writing a procedure:

```
DTR> READY PHONES
DTR> DEFINE PROCEDURE PHONES_REPORT
DFN> REPORT PHONES ON *."device or file"
DFN> SET REPORT_NAME = "PHONES LIST"
DFN> SET COLUMNS_PAGE = 50
DFN> PRINT F!!!!, AREA!!!!NUM
DFN> END_REPORT
DFN> END_PROCEDURE
DTR> SHOW PROCEDURES
Procedures:
    PHONES_REPORT;1
```

To execute a DATATRIEVE procedure, type a colon (:) followed by the name of the procedure, then press the RETURN key. When you execute PHONES_REPORT, DATATRIEVE prompts you for where you want the report. If you enter TT: in response to the prompt, DATATRIEVE displays the report on your terminal:

```
DTR> :PHONES_REPORT
Enter device or file: TT:
```

```
                PHONES LIST          27-Feb-1984
                                      Page 1

LISA BELL          603 883-8275
PHYLLIS CLERC     603 884-9907
JANE LINTE        603 884-5678
BONNIE SCHUTZ    617 567-8712
JOHN WINTLOW     205 880-6789
```

```
DTR>
```

If you execute the procedure again and type the name of a printer device, such as LP:, in response to the prompt, DATATRIEVE prints the report on your line printer. If you respond with the name of a file (PHONES.LIS, for example), DATATRIEVE creates the report as a file stored in your VMS directory.

If you made any mistakes when defining the PHONES_REPORT procedure, DATATRIEVE displays one or more error messages at the time you execute it. If this happens, you might notice that it continues to process the statements and commands left in the procedure after the first error occurs. (Chapter 17 tells you how to use the SET ABORT command to prevent this from happening.)

If you simply type EDIT and press the RETURN key, you can edit only the last statement or command DATATRIEVE executed for you. To correct the errors in the procedure, you must follow the EDIT command with the procedure name. After you make your corrections and before you exit the editor, double check the lines in your procedure against those in the example. Then exit the editor and execute the procedure again. See the chapter on editing for more information about the editors available to you with DATATRIEVE.

1.8 What Do I Read Next?

The sections of this book you should read next depend on your level of experience with the VMS operating system and computer languages and also on how much guidance you expect when learning something new:

- If you are unfamiliar with the VMS operating system and have limited experience writing programs in computer languages, you can:
 - Read Chapters 2 and 3 to learn about the VMS operating system and the editors you can use with DATATRIEVE.
 - Read Chapter 6 to find out how you can get online assistance when using DATATRIEVE
 - Go on to Chapters 8 through 20 to read more about creating your own applications with DATATRIEVE

After you get more DATATRIEVE experience, you can read Chapter 4 to get some help with syntax diagrams, Chapter 5 to learn about startup command files and input entry, and Chapter 7 to learn more details about managing CDD directories and their contents.

- If you are familiar with the VMS operating system, but have limited programming experience, you do not need to read Chapters 2 and 3. Otherwise, follow the course just outlined.
- If you are an experienced programmer, read the next section. You can then continue reading chapters in the order that they appear in the book, skipping Chapters 2 and 3 if you are already familiar with the VMS operating system.
- Feel free to read whatever intrigues you and then start right in using DATATRIEVE. Use the index or table of contents to look up additional information.

1.9 What DATATRIEVE Can Do for the Programmer

This section is for readers who can use one or more programming languages, such as COBOL or BASIC. If you fall into this category, you may want additional information on how DATATRIEVE is different from the languages you have used before.

DATATRIEVE provides the same data storage capabilities that you have with other languages. It can store and retrieve data using existing data files of any type that are supported by VAX Record Management Services (RMS). It can also create sequential and multikey indexed files, but not relative files.

DATATRIEVE is an excellent query and report writing interface for databases created and maintained by VAX DBMS or the VAX relational database products. This book does not contain information on using DATATRIEVE with DBMS or relational databases. Refer to the *VAX DATATRIEVE User's Guide* if you are interested in this subject.

In COBOL or BASIC, each program describes the structure of input and output records. DATATRIEVE lets you define records and store record definitions separately from the procedures that use them. You can then write any number of procedures that use the records you have defined, without redefining the record each time.

DATATRIEVE also lets you create data definitions, called **view domains**, that can access either a subset of the fields in one data file or a combination of fields from more than one file. View domains can help you reduce the number of statements that you have to write when retrieving data.

DATATRIEVE also handles other common language functions automatically, without the need for language statements. For instance, DATATRIEVE:

- Finds data files, opens them, and performs input/output operations
- Labels columns in an output display
- Converts data types
- Formats data for output
- Handles conditions like EOF and matching

As a result, you can save many lines of code, get applications running quickly, and still have code that is more readable than COBOL or BASIC.

DATATRIEVE does not give you all the options available with other languages:

- DATATRIEVE lets you set up data hierarchies such as the repeating fields generated by a COBOL OCCURS clause, although retrieving data from repeating fields is not as easy as retrieving data from other types of fields. DATATRIEVE does not have a system of subscripts or indexes that let you explicitly specify an occurrence in a repeating field. Be sure you consider this fact before you decide to use the DATATRIEVE OCCURS clause.
- The DATATRIEVE language does not contain clauses such as BLOCK SIZE and CONTIGUOUS BEST TRY that let you optimize files for best response time. If you are setting up or maintaining large data files, therefore, you should use the utilities provided by VAX RMS to load and maintain these files. Refer to the chapter on optimizing DATATRIEVE performance in the *VAX DATATRIEVE User's Guide* if you want more information on this subject.

- DATATRIEVE procedures are not compiled when they are stored. Every time you execute a DATATRIEVE procedure, DATATRIEVE processes each statement or command in sequential order, just as if you were entering each one interactively. The advantage in using procedures, therefore, is more a matter of convenience than speed of execution.

By using the DATATRIEVE Call Interface, you can include DATATRIEVE functions in a program written in another language. The Call Interface is often used in two ways:

- You can use the linkage section of a COBOL program to do file access entirely through DATATRIEVE. In this way, the calling program does not need to specify the structure of the data, and you do not need to relink programs when the data files change.
- You can write a program that passes commands and statements to DATATRIEVE. The program can present the user with a customized interface, such as a menu. In this way, you can “hide” DATATRIEVE from users who do not know how to use its commands and statements.

The *VAX DATATRIEVE Guide to Programming and Customizing* explains how you can use DATATRIEVE with other languages and provides more information about how you can customize DATATRIEVE features for your installation.



VMS Concepts 2

As a DATATRIEVE user, knowledge of the VMS operating system is very important. You gain access to DATATRIEVE through the VMS system and the data files you create are stored in VMS directories.

This chapter describes how to:

- Use the DIGITAL Command Language (DCL) to create and manipulate files
- Log in
- Protect your files and directories
- Set up a useful login command file (LOGIN.COM) to execute commands automatically
- Use the DTR\$STARTUP logical to create a startup command file

2.1 Using DIGITAL Command Language (DCL)

An operating system is the system software that controls the operations of the computer. In this chapter, the operating system is referred to simply as the system. To perform data processing operations on a VMS system, you issue instructions in the DIGITAL Command Language (DCL). Like any language, DCL consists of a vocabulary and rules of grammar.

The vocabulary of DCL includes commands, parameters, and qualifiers. These perform functions similar to those of verbs, nouns, adverbs, and adjectives in English. When arranged to form a command string, these words describe to the system the operation you want to perform.

2.2 Logging In

To begin a session at your terminal, you must log in. Logging in consists of getting the system's attention and identifying yourself as an authorized user.

Before you can log in to the system, however, you need an account. Accounts are set up by the system manager, or whoever is responsible at your installation for authorizing the use of the system. This person can provide you with a user name and password.

Your user name is a unique name that identifies you to the system and distinguishes you from other users. Your password is for your protection. If you maintain its secrecy, other users cannot use system resources under your name or gain access to files you want to keep private.

When you log in, you must enter both your user name and password before VMS lets you begin typing commands.

2.2.1 Getting the Terminal Ready

Before you use the terminal, be sure that:

- The terminal is plugged in and the power is turned on.
- If the terminal has a LOCAL/ON LINE switch, the switch is set to ON LINE. When it is on line, the terminal communicates with your system. When it is set to LOCAL, the terminal is electrically disconnected from your system. The following list explains how to set your terminal to ON LINE:
 - On a VT100 series terminal, press the 4 key on the main keyboard while in Set-Up mode A.
 - On a VT200 series terminal, press the Set-Up key to access the Set-Up Directory. In the Set-Up Directory, the ON LINE/LOCAL option is located in the first screen, immediately below the DISPLAY option. If the option is currently ON LINE, you do not need to change anything. If the option is currently LOCAL, move your cursor to the LOCAL option using the down arrow key. Change the option from LOCAL to ON LINE by pressing the keypad Enter key. Exit the Set-Up Directory by pressing the Set-Up key again.
 - If you are using a dialup connection, check installation instructions for special procedures.

If you have any problems with the login procedure described in the next section, get help from the system manager. There are several reasons your login may not work. Among them are:

- The terminal may not be properly connected to the computer
- The baud rate (speed at which the terminal transmits or receives characters) may not be correctly set

2.2.2 Gaining Access to the System

Press the RETURN key or CTRL/Y to signal the system that you want to log in. The system responds by prompting you for your user name. Enter your user name and press the RETURN key. The system displays your user name as you type it. After you enter your user name and press the RETURN key, the system prompts you to enter your password. When you type the password, the system does not display it (in order to preserve its secrecy.) The login sequence looks like this:

```
Username: BELL
Password:
```

```

                Welcome to VAX/VMS version V4.5 on node YOURNODE
Last interactive login on Thursday, 11-JUN-1987 16:09
Last non-interactive login on Thursday, 11-JUN-1987 10:46
                You have 5 new Mail messages.
```

```
$
```

Some systems print a welcoming message above the username prompt. If your system is part of a network, your system name (node) will be printed after the VMS version number.

The dollar sign (\$) is a symbol the system uses as a prompt. It is the VMS default system prompt. If you or your system manager change the default with the DCL SET PROMPT command, you see the changed prompt instead. When the system displays this character at the left side of your screen, it indicates that the login was successful and that you can begin entering commands.

If you type your user name or your password incorrectly, the system displays an error message. When an error message appears, you must repeat the entire login procedure.

2.3 Getting Online Help

When you use the VMS operating system, you may not always have a reference manual available at your terminal. You may want to get some help on a subject that you need to know more about. The HELP command is designed to provide you with this information. For example, to display a list of topics for which help is available, type:

```
$ HELP
```

The system displays the list of topics and then prompts for your choice. If you want information about a particular subject, type its name after the prompt. For instance:

```
Topic? PRINT
```

The information displayed includes a synopsis of what the PRINT command does, the parameters it requires, and the qualifiers it can take.

If you want to know more about one of the PRINT commands qualifiers, respond to the prompt PRINT subtopic? with that qualifier. For example, to display information about the /COPIES qualifier of the PRINT command, type:

```
PRINT subtopic? /COPIES
```

If you already know the topic and even the subtopic on which you need help, you can simply type, for example:

```
$ HELP PRINT/COPIES
```

2.4 Entering Commands

All commands to the system are words, generally verbs, that describe the functions they perform. You can type them in uppercase or lowercase. For example:

```
$ SHOW TIME
```

The system responds to this command by displaying the current date and time:

```
29-NOV-1985 09:25:07
```

Command parameters define what the command acts upon; command qualifiers further define how that action occurs. For instance, the following PRINT command requires an object, or parameter, to indicate what is to be printed:

```
$ PRINT BILLING.LIS
```

In this command, BILLING.LIS is a parameter for the PRINT command, indicating the name of the file to be printed. A space separates the command and its parameter.

Command qualifiers restrict or modify the function that the command is to perform. For example:

```
$ PRINT/COPIES=2 BILLING.LIS
```

In this command, /COPIES=2 is a qualifier that indicates how many copies of the file BILLING.LIS you want printed. A slash character (/) precedes the qualifier.

The rules of grammar for DCL (that is, the order of the words, the spacing, and the punctuation) are also strictly defined. Your system documentation set contains a dictionary of DCL commands and discusses the rules of grammar.

2.4.1 Command Prompting

When you enter a command at the terminal, you do not need to enter the entire command. If you enter a command without specifying required parameters, or you forget what comes next, the system prompts you for the additional information:

```
$ PRINT  
$_File:  BILLING.LIS
```

In this example, no parameter followed the PRINT command, so the system prompted for the name of the file to be printed.

If a command requires two or more parameters, it prompts for each parameter:

```
$ COPY  
$_From:  OLDFILE.TXT  
$_To:    NEWFILE.TXT
```

You can enter both file names after the first prompt:

```
$ COPY  
$_From:  OLDFILE.TXT NEWFILE.TXT
```

You can also enter the entire command line on one line:

```
$ COPY OLDFILE.TXT NEWFILE.TXT
```

2.4.2 Defaults

A **default** is the value supplied by the operating system when you do not specify one yourself. For instance, if you do not specify the number of copies as a qualifier for the PRINT command, the system uses the default value of 1. By not explicitly stating a choice, you imply the default. VMS supplies default values for many commands. The defaults used with individual commands are specified with each command's description in both online help and in the DCL documentation included in your system's documentation set.

2.4.3 Abbreviating Commands

You do not always need to type the full command. You must type at least the minimum number of characters necessary to identify the command uniquely.

For example, the SET, SEARCH, and SHOW commands all begin with the letter S. To identify the SHOW command, you must type at least two characters, SH. To identify the SET and SEARCH commands, you must type three characters, SET and SEA respectively.

The examples in this chapter show full commands, so that you can become familiar with the commands and what they do.

2.4.4 Recovering from Errors

If you are entering a command and you make a mistake, you can use the following keys to correct the mistake or abort the command:

- **DELETE**

Backspaces over one character typed on the current line, then deletes the character. Most video display terminals actually move the cursor (an underline or block that marks your position) backward and erase the character when you press the DELETE key.

- **CTRL/U**

Ignores the current line and performs a return so you can reenter the entire line. Use CTRL/U when a line contains a number of mistakes and it would be tedious to use the DELETE key.

- CTRL/Z or CTRL/Y

Cancels an entire command, regardless of how many lines were used to enter it.

You can also use CTRL/Y or CTRL/C to interrupt the system while it is executing a command. Press CTRL/Y (or CTRL/C) and the system terminates the current process and returns you to DCL level:

```
$ TYPE BILLING.LIS
```

```
.
```

```
.
```

```
CTRL/Y
```

```
$
```

In this example, CTRL/Y interrupted the typing of a long file and returned to DCL level.

- CTRL/B or Up Arrow

Displays previously entered commands so that you can reproduce them. When you press CTRL/B or the up arrow key, the previous command is displayed. (You can recall up to 20 previously entered commands.) Press the down arrow key to display commands in the other direction. You can edit the command string by pressing the left and right arrow keys, which move the cursor. You can also move the cursor to the beginning of the line by pressing CTRL/H and to the end of the line by pressing CTRL/E. Then, you can overstrike the character you want to change. Or, you can press CTRL/A, which lets you insert a character rather than overstrike it.

- HOLD SCREEN (F1) or NO SCROLL

Suspends and resumes the scrolling, or upward movement, of the terminal display. To temporarily stop the display from scrolling, press the HOLD SCREEN (F1) key on a VT200-series terminal. On a VT100-series terminal, press the NO SCROLL key. To continue scrolling, press the key again.

CTRL/S suspends the VT52 terminal display of the file and the processing of the command. To resume display, press CTRL/Q. The interrupted command displays lines beginning at the point at which processing was interrupted.

2.4.5 Summary of Entering Commands

The following list summarizes the rules you must follow when entering commands to the VMS operating system:

- You must precede each qualifier name with a single slash character (/).
- If you omit a required parameter (for example, a file specification), the DCL command interpreter prompts you to enter it.

- You can truncate any command name or qualifier name to four characters. Fewer than four characters are acceptable, as long as there is no ambiguity about the name.
- After you have entered a complete command, you must press the RETURN key to pass the command to the system for processing.
- You can cancel a command before the final return by using CTRL/Y.
- You can interrupt command execution by using CTRL/Y. To resume the interrupted command, enter the CONTINUE command. To stop processing completely after using CTRL/Y, you can begin entering other DCL commands.

2.5 Interpreting System Responses

When you enter a command, the system can:

- Execute the command, indicating successful completion with the dollar sign prompt
- Execute the command and inform you in a message of what it has done
- Inform you of errors you have made, if execution is not successful

2.5.1 Information Messages

The system responds to some commands by giving you information about what it has done. For example, when you use the PRINT command, the system displays the job identification number it assigned to the print job and shows the print queue the job has entered:

```
$ PRINT BILLING.LIS
  Job 500 entered on queue SYS$PRINT
```

Not all commands display informational messages; in fact, successful completion of a command is most commonly indicated by a dollar sign prompt for another command. Unsuccessful attempts are always indicated by one or more error messages.

2.5.2 Error Messages

If you enter a command incorrectly, the system displays an error message and prompts for a new command line. It ignores the incorrect command:

```
$ CAPY
%DCL-W-IVVERB, unrecognized command
  \CAPY\
$
```

The code preceding the text of the message indicates that:

- The message is from DCL, the command interpreter
- It is a warning (W) message
- The mnemonic for this particular message is IVVERB

You can also receive error messages during command execution if the system cannot perform the function you have requested. For example, if you type a PRINT command correctly, but the file that you specify does not exist, the PRINT command informs you of the error:

```
$ PRINT PAYROLL.DAT
%PRINT-W-OPENIN, error opening DBA1:[BELL]PAYROLL.DAT; as input
-RMS-E-FNF, file not found
```

The first message is from the PRINT command. It tells you it cannot open the specified file. The second message indicates the reason, that is, the file cannot be found. RMS refers to the VMS file-handling facility, Record Management Services; error messages related to file handling are generally VAX RMS messages.

2.6 Logging Out

When you have finished using the computer, use the LOGOUT command to end the terminal session:

```
$ LOGOUT
BELL logged out at 30-MAY-1984 15:30:10
```

Note that neither shutting off your terminal nor setting the ON LINE/LOCAL switch to LOCAL automatically causes you to log out. To ensure that you have logged out, you should use the LOGOUT command to end a terminal session. If you shut a terminal off without logging out properly, another user may be able to turn the terminal on later and use your account.

2.7 File Management

This section explains how to:

- Create, identify, delete, and purge files
- Display and print files
- List files in a directory
- Copy and rename files

- Append files to other files
- Find differences between files
- Search files for a specified string

2.7.1 Creating Files

You can create files in two ways. Chapter 3 tells you how to create a file by using the EDIT command. You can also use the CREATE command to make a new file. Specify the file name as a parameter. You can insert text immediately and end with CTRL/Z:

```
$ CREATE NEWFILE.TXT
THIS IS THE FIRST LINE. CTRL/Z
^Z
$
```

You cannot use the CREATE command to modify an existing file.

2.7.2 Identifying Files

A complete file specification contains all the information the system needs to locate and identify a file. A complete file specification has the format:

```
node::device:[directory]filename.type;version
```

The punctuation marks (colons, brackets, period, semicolon) are required syntax that separate the various components of the file specification. For example:

```
BOOKIE::DBA2:[BELL]PERSONNEL.DAT;5
```

2.7.2.1 Nodes — When computer systems are linked together to form a network, each system in the network is called a node and is identified within the network by a unique node name. Your system may or may not be part of a larger network.

If your system is a network node, you can gain access to a file located at another node on the network by adding a node specification to the first part of the file specification. This specification lets you access the file only if the owner of the file has permitted other users access to it. If you do not specify a node, the system assumes by default that the file belongs to your own node.

2.7.2.2 Devices — The second part of a file specification, the device name, identifies the physical device (for example, a disk) on which a file is stored. As a DATATRIEVE user, your data is most likely to be stored on disks.

If you omit a device name from a file specification, the system supplies a default value; that is, it assumes the file is on the disk assigned you when the system manager set up your account. This is your default disk.

2.7.2.3 Directories and Subdirectories — Because a disk can contain files belonging to many different users, each user of a given disk has a directory that catalogs all the files belonging to him on that device.

Directories and subdirectories are discussed in greater detail later in this chapter.

2.7.2.4 File Names, Types, and Versions — By taking advantage of your default node, disk, and directory, you can identify a file uniquely by specifying only its file name and file type in the format:

filename.typ

The file name can have from zero to 39 characters chosen from the letters A through Z and the numbers 0 through 9. When you create files, give them names that are meaningful to you.

The file type can be from zero to 39 characters and must be preceded by a period. Again, you can choose any of the letters A through Z or the numbers 0 through 9 for the file type. The file type usually describes default file types used for special purposes.

Table 2-1 lists some of the default file types.

Table 2-1: Default File Types

File Type	Use
DAT	Data file
EDT	Startup command file for EDT editor
EXE	Executable program image file
JOU	Journal file used by the EDT editor

(continued on next page)

Table 2-1: Default File Types (Cont.)

File Type	Use
LIS	Output listing file
MAI	Mail message file
OBJ	Object module file output from a compiler or assembler
TJL	Journal file use by the LSE and VAXTPU editors

In addition to a file name and type, every file has a version number that the system assigns when the file is created or revised. The original number is one. When you create additional versions of the file, the version number is automatically increased by one.

You rarely need to specify the version number with a file specification. The system assumes default values for version numbers, as it does with devices, directories, and file types. Version number defaults are determined as follows:

- For an input file, the system uses the highest existing version number of the file.
- For an output file, the system adds one to the highest existing version number.

When you use a version number in a file specification, precede the version number with a semicolon (;).

2.7.2.5 Wildcard Character — A wildcard character is a symbol that you can use with many DCL commands to apply the command to more than one file, rather than specifying each file individually. The asterisk (*) can be used in a specification of a directory, file name, file type, and version number.

For example, you can specify all versions of a file by using an asterisk in place of the version number in the file specification. If, for example, you want to print all versions of the file STARTUP.COM type:

```
$ PRINT STARTUP.COM;*
```

If there were no wildcard character in the previous example, the PRINT command by default would apply only to the most recent version of the file STARTUP.COM.

The following command prints all versions of all files in the current directory with the file type of .COM:

```
$ PRINT *.COM;*
```

To print all versions of all files in the directory with the file name of STARTUP, type:

```
$ PRINT STARTUP.*;*
```

2.7.3 Deleting Files

The DELETE command deletes specific files. When you use the DELETE command, you must specify a file name, file type, and version number. This provides some protection against accidental deletion. However, you can specify any of these file components as a wildcard character. You can also enter more than one file specification on a command line, separating the file specifications with commas. Some examples of the DELETE command are shown in Table 2-2.

Table 2-2: Examples of the DELETE Command

Command	Result
\$ DELETE AVERAGE.OBJ;1	Deletes the file named AVERAGE.OBJ;1
\$ DELETE .LIS;*	Deletes all files with file types of LIS
\$ DELETE A.DAT;1,A.DAT;2	Deletes the first two versions of the same data file

2.7.4 Purging Files

You may want to clean up your directory by getting rid of early versions of particular files. If you have many versions of a file, naming them all in the DELETE command would be tedious.

The PURGE command allows you to delete all but the most recent version of a file; therefore, no version number is allowed with the PURGE command. For example:

```
$ PURGE AVERAGE.FOR
```

This command deletes all files named AVERAGE.FOR, except the file with the highest version number.

The `/KEEP` qualifier of the `PURGE` command allows you to specify that you want to keep more than one version of a file. For example:

```
$ PURGE/KEEP=2 TEST.COM
```

This command deletes all but the two most recent versions of the file `TEST.COM`.

2.7.5 Displaying Files at Your Terminal

The `TYPE` command displays a file at your terminal. For example:

```
$ TYPE ZAPATA.LIS
```

```
EVEN IF YOU KILL ZAPATA,  
THERE WILL BE OTHERS  
TO TAKE HIS PLACE.
```

While a file is being displayed, you can interrupt the output by using `CTRL/C` or `CTRL/Y`. The system then prompts you to enter another command.

2.7.6 Printing Files

When you use the `PRINT` command to obtain a printed copy of a file, the system cannot always print the file immediately, because other users may be printing files. The system enters the name of the file you want to print in a queue and prints the file at the first opportunity.

A printed file is preceded by a header page describing the file so you can identify your own listing. For example, if you issue the following command, the header page will show your user name and the file name, type, and version number of the file:

```
$ PRINT DB2:[BELL]AVERAGE.LIS  
Job 435 entered on queue SYS$PRINT
```

When you use the `PRINT` command, the system responds with a message indicating the job number it assigned to the print job.

The `PRINT` command also has qualifiers that allow you to control the number of copies of the file to print, the type of forms to print the file on, and so on. You can find more information on these qualifiers in the VMS documentation for the Digital Command Language.

2.7.7 Listing Files in a Directory

The `DIRECTORY` command lists the names of files in a particular directory. If you type the `DIRECTORY` command with no parameters or qualifiers, the command displays the files listed in your default directory. For example:

```
$ DIRECTORY
DIRECTORY DBA2:[BELL] (1)
LOGIN.COM;45 PERSON.DAT;13 PET.DAT;3 PHONES.DAT;4 (2)
YACHTS.DAT;67
Total of 5 files. (3)
```

Note that the `DIRECTORY` command shows:

1. The disk and directory name
2. The file name, file type, and version number of each file in the directory
3. The total number of files in the directory

When you enter the `DIRECTORY` command, you can provide one or more file specifications to obtain a listing about particular files only. For example, to find out how many versions of the file `AVERAGE.FOR` currently exist, issue the `DIRECTORY` command as follows:

```
$ DIRECTORY AVERAGE.FOR
DIRECTORY DBA2:[BELL]
AVERAGE.FOR;2 AVERAGE.FOR;1
Total of 2 files.
```

There are many helpful qualifiers to the directory command. They give you information such as the date and time a file was created, the size of the file, and the owner and protection of the file. Refer to the VMS documentation for online help for more information about using this versatile command.

2.7.8 Copying Files

The `COPY` command makes copies of files. You can use it to make copies of files in your default directory, to copy files from one directory to another directory, to copy files from other devices, or to create files consisting of more than one input file.

When you issue the COPY command, you specify first the name of the input file you want to copy, then the name of the output file. For example, the following COPY command copies the contents on the file PAYROLL.TST to a file named PAYROLL.OLD:

```
$ COPY PAYROLL.TST PAYROLL.OLD
```

If a file named PAYROLL.OLD exists, a new version of that file is created with a higher version number.

You can copy a file from the directory [BELL] to the subdirectory [BELL.TESTFILES] and give it a new name, OLDFILE.DAT:

```
$ COPY NEWFILE.DAT [BELL.TESTFILES]OLDFILE.DAT
```

When you copy files from devices other than your default disk, you must specify the device name in the COPY command. For example:

```
$ COPY DBA1:[JONES]PET.DAT;2 DBA2:[BELL]PET.DAT
```

2.7.9 Renaming Files

The RENAME command changes the identification of one or more files. For example, the following command changes the name of the most recent version of the file PAYROLL.DAT to TEST.OLD.

```
$ RENAME PAYROLL.DAT TEST.OLD
```

You can use the RENAME command to move a file from one directory to another. For example, the following command moves TEST.OLD from the directory [MALCOM] to the subdirectory [MALCOM.TESTFILES]:

```
$ RENAME [MALCOM]TEST.OLD [MALCOM.TESTFILES]
```

You can use wildcard characters if you want to change a number of files that have either a common file name or file type. For example:

```
$ RENAME PAYROLL.*.* [MALCOM.TESTFILES]*.*.*
```

This RENAME command changes the directory name for all versions of all files that have file names of PAYROLL. The files are now cataloged in the subdirectory [MALCOM.TESTFILES].

You cannot use the RENAME command to move files from one disk to another.

2.7.10 Appending Files

The APPEND command adds the contents of one or more input files to the end of a specified output file. For example:

```
$ APPEND TEST.TXT NEWTEST.TXT
```

The APPEND command appends the contents of the file TEST.TXT from the default directory to the end of the file NEWTEST.TXT.

2.7.11 Finding Differences Between Files

The DIFFERENCES command compares the contents of two files and creates a listing of those records that do not match. For example:

```
$ DIFFERENCES PAYROLL.DAT:2 PAYROLL.DAT:1
```

The DIFFERENCES command compares the contents of PAYROLL.DAT:2 and PAYROLL.DAT:1 in the current default directory. By default, DIFFERENCES compares every character in every record and displays the results on the terminal.

Several qualifiers are available which let you modify the format of the information produced, control the extent of the comparison, and ignore selected data in each record.

2.7.12 Searching Files for a Selected String

The SEARCH command searches one or more files for a specified string or strings and lists all the lines containing occurrences of the string. For example:

```
$ SEARCH BRANDO.TXT,WILLIAMS.TXT STELLA
```

The SEARCH command searches the files BRANDO.TXT and WILLIAMS.TXT for occurrences of the character string STELLA. Each line containing the string is displayed at the terminal.

As with the DIFFERENCES command, several qualifiers are available to modify the SEARCH command.

2.8 Creating and Managing VMS Directories

The main reason for creating directories and subdirectories is to separate information logically on a disk. When users are separated from one another through the use of top-level directories, each user appears to own a portion of the disk for storage of information. The system also supports protection of the directories, which can be used to prevent other users from accessing files. This protection can be used to protect an entire directory from access or to protect only a few of the files in a directory.

In some situations, one user could be working on several projects, each requiring several files. Subdirectories can be used to separate the files belonging to one project from files belonging to another.

Subdirectories become useful for a frequent user because directory listings can be very long. When information is separated, each directory is smaller and easier to work with.

2.8.1 Directory Structure

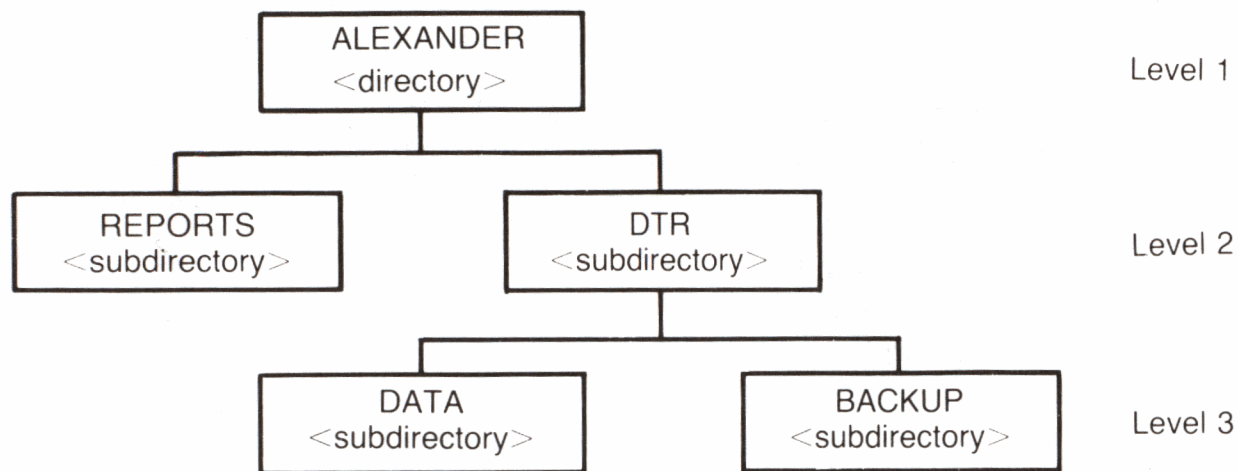
A **directory file** is a special kind of file. It contains a list of names of other files. The system uses the directory to access these files. Directories reside on disk volumes, with one directory file generally created by the system manager for each user.

Your main directory (also called the top-level directory) resides in the master file directory (MFD) on the disk volume. This MFD catalogs all user file directories (UFD). Your main directory is one of many UFDs, and it lists all of your personal files and subdirectories. These subdirectories may contain a list of file names, some of which may be other subdirectories.

These rules apply to directories and subdirectories:

- Although the name of the directory is listed as `directory.DIR`, you must specify it as `[directory]` in the file specification.
- Directory files are always version 1.
- The number of directory files that may be listed in any directory file is limited to the amount of disk space available to you. For example, `SMITH.DIR` could contain the names of more than one subdirectory, and each subdirectory file could contain the names of several other subdirectory files.
- You can have as many as eight levels of directories.

Figure 2-1 shows a sample directory structure for a user named Alexander.



ZK-00001-00

Figure 2-1: Sample VMS Directory Structure

2.8.2 Accessing Other Directories

You can gain access to files in other directories (including directories that catalog files belonging to other users) by specifying the directory name in a file specification. For example, to display on your terminal the contents of a file named CONTENTS.LIS belonging to a user whose directory is [BRANDO], issue the TYPE command:

```
$ TYPE [BRANDO]CONTENTS.LIS
```

Note that the file specification does not include a device name. For this command to execute successfully, the directory [BRANDO] must be on your default disk device. This is because the system always applies a default when you omit a device name. If user Brando's directory is on the disk DBB2, you would issue the TYPE command as:

```
$ TYPE DBB2:[BRANDO]CONTENTS.LIS
```

For either of the previous examples to work correctly, Brando must have given other users access to files in the directory. You can explicitly allow or restrict access to your own files, either generally or on a file-by-file basis, with the SET PROTECTION command. You will see how to do this later in this chapter.

2.8.3 Creating Subdirectories

The CREATE/DIRECTORY command creates a subdirectory. For example:

```
$ CREATE/DIRECTORY [BELL.REMINDER]
```

This command creates the subdirectory file, REMINDER.DIR, in the directory [BELL]. You can specify the subdirectory name, [BELL.REMINDER], in commands or programs.

Files can also be cataloged in subdirectories. A subdirectory is a file (cataloged in a higher directory) that contains additional files. A subdirectory name is formed by concatenating its name to the name of the directory that lists it. For example:

```
$ TYPE [BRANDO.PLAYS]DESIRE.DAT
```

This TYPE command requests a display of the file DESIRE.DAT that is cataloged in the subdirectory [BRANDO.PLAYS]. The subdirectory file name is PLAYS.DIR and is cataloged in the directory [BRANDO].

2.8.4 Changing Your Default Directory

As with the default disk, if you do not specify another directory, or if you do not specify any directory, the system applies the default. It assumes that the files to which you refer are cataloged in your default directory. You can find out what your current default disk and directory are by issuing a SHOW DEFAULT command:

```
$ SHOW DEFAULT
```

```
DBA2:[BELL]
```

This response from the SHOW DEFAULT command indicates that the user's default device is DBA2 and the default directory is [BELL].

To establish another directory or subdirectory as your default directory, use the SET DEFAULT command. For instance, you can set the default to [BELL.REMINDER] and then issue the DIRECTORY command:

```
$ SET DEFAULT [BELL.REMINDER]  
$ DIRECTORY
```

```
Directory DBA2:[BELL.REMINDER]
```

```
MEMO.TXT;1  REPORT.DAT;1
```

```
Total of 2 files
```

```
$
```

You are now working out of this directory, and any new file you create is cataloged in the subdirectory [BELL.REMINDER]. You could also do this by specifying the subdirectory as part of the file specification when you use the EDIT command.

You can change your default directory as often as is convenient. The latest change you make with the SET DEFAULT command remains in effect until you either issue another SET DEFAULT command or log out.

2.8.5 Protecting Your Directories and Files

The VMS operating system protects data on directories and in files to ensure against accidental or unauthorized access. Protection is provided by means of an owner user identification code (UIC). The owner UIC is divided into a group name and a user name. For example:

```
$ DIRECTORY/OWNER PET.DAT
Directory DBA2:[BELL]
PET.DAT;13          [STUDENTS,BELL]
Total of 1 file, 1 block.
$
```

The protection code indicates that BELL has a group UIC name of STUDENTS, which is displayed before his user name.

When a user attempts to access a directory or file, his UIC is compared to the owner UIC. Depending on the relationship of the UICs, the user is in one or more of the following categories:

- System – These group numbers are generally reserved for system managers, system programmers, and operators.
- Owner – The user with the same UIC as the person who created and therefore owns the directory or file.
- Group – All users, including the owner, who have the same group number in their UICs as the owner of the subdirectory or file.
- World – All users, including those in the first three categories.

Each of these categories of user can be granted or denied any of the following types of access:

- Read – The right to examine, print, or copy a directory or file.
- Write – The right to modify a file or to write files onto a disk.

- Execute – The right to execute files that contain executable program images.
- Delete – The right to delete the directory, file, or files.

When you specify a user category with a protection code of 0, or null code, you deny that user category any access.

The system provides a default protection code for directories and files you create. You can determine the current default protection by issuing the SHOW PROTECTION command:

```
$ SHOW PROTECTION
SYSTEM=RWED,OWNER=RWED,GROUP=RE,WORLD=NO ACCESS
```

This response is the system default protection. It indicates that the system and the owner have all types of access, members of the owner's group have read and execute access, and all other users (the world) have no access.

When you create a directory or file, you can define the protection you want to be applied if you do not want to use the default protection. For example:

```
$ SET PROTECTION=(S:RWED,O:RWED,G:RE,W:R) PAYROLL.DAT
```

The SET PROTECTION command in this example allows the system and owner read, write, execute, and delete privileges; allows the group read and execute privileges only; and restricts the world to read privileges for the file PAYROLL.DAT.

To determine the current protection associated with a specific directory, file or files, use the /PROTECTION qualifier on the DIRECTORY command. For example:

```
$ DIRECTORY/PROTECTION PERSONNEL.RNO
DIRECTORY DBA1:[BELL]
PERSONNEL.RNO;5      (RWED,RWED,RW,R)
```

Total of 1 file, 8 blocks.

You can change the default protection with the SET PROTECTION/DEFAULT command. This command indicates that the protection code you specify is to be applied to all directories and files that you subsequently create during the terminal session. For example:

```
$ SET PROTECTION=(S:RWED,O:RWED,G:R,W)/DEFAULT
```

This command sets the default protection to give the system and owner unlimited access, give the group read access and give no access to the world.

The directory protection can override the protection of individual files in the directory. For example, if a directory denies world access, world users cannot look at even those files in the directory that permit world access. To guarantee protection, however, individual files must also be protected.

2.8.6 Deleting a Directory

The VMS system provides two safeguards against the accidental deletion of a directory. You must delete all files in a subdirectory and then change the protection of the subdirectory from the default provided by the system. For example, the SET DEFAULT command sets the default to the directory targeted for deletion:

```
$ SET DEFAULT [BRANDO.PLAYS.WILLIAMS]
```

The DELETE command deletes all files in the subdirectory:

```
$ DELETE *.*;*
```

This SET DEFAULT sets the default to the directory containing the subdirectory WILLIAMS.DIR:

```
$ SET DEFAULT [BRANDO.PLAYS]
```

The SET PROTECTION changes the default protection to allow deletion:

```
$ SET PROTECTION=0:D WILLIAMS.DIR
```

The subdirectory is then deleted:

```
$ DELETE WILLIAMS.DIR;
```

2.9 Logical Names

An alternate way of referring to a specific device, directory, or file is to use a logical name. Two important reasons for using logical names are to:

- Achieve device and file independence
- Reduce typing of long file specifications

A logical name can represent an entire file specification or the leftmost portion of one. You can create logical names with the ASSIGN command. For example:

```
$ ASSIGN DBA2:[BELL.TESTFILES] TEST  
$ TYPE TEST:MEMO.LIS
```

The ASSIGN command creates the logical name TEST to represent the directory specification DBA2:[BELL.TESTFILES]. When TEST is used in the TYPE command, the system translates it. The logical name in the file specification is replaced by its current equivalence name. The TYPE command in the previous example displays the file DBA2:[BELL.TESTFILES]MEMO.LIS.

Only one logical name is permitted in a file specification. It must be the first or only element of the file specification, and it must be followed by a colon if any other element is present.

The VMS system maintains tables of all logical names that are created by users. By default, you use logical names from one of these four tables:

- Process logical name tables. A separate logical name table exists for every user, or process, on the system. Names in a process logical name table are available only to the user who defines them. An ASSIGN command places a logical name in your process logical name table by default.
- Job logical name tables. The job table contains logical names that are available to your process and any of your subprocesses.
- Group logical name tables. A separate logical name table exists for every group in the system. The names in any of these tables can be accessed only by users who have the same group number in their user identification code.
- System logical name table. There is a single system logical name table. The logical names in this table can be accessed by all users.

2.10 System Default Logical Names

When you log in, the system provides several default logical names. These names are used by the command interpreter to read your commands and to print responses or error messages. Table 2-3 describes the default logical names.

Table 2-3: System Default Logical Names

Logical Name	Use
SYS\$INPUT	<p>The default input stream from which the system reads commands and your programs read data.</p> <p>Default interactive assignment: your terminal.</p> <p>Default batch assignment: the command procedure or batch stream.</p>
SYS\$OUTPUT	<p>The default output stream to which the system writes responses to commands and your programs write data.</p> <p>Default interactive assignment: your terminal.</p> <p>Default batch assignment: batch job log file.</p>
SYS\$ERROR	<p>The default device to which the system writes all error and informational messages.</p> <p>Default interactive assignment: your terminal.</p> <p>Default batch assignment: batch job log file.</p>
SYS\$DISK	<p>Your default disk device.</p> <p>Default assignment: set in user authorization file.</p>

By default, when the system translates a logical name, it searches the process, job, group, and system logical name tables, in that order. (There are ways to change the order of this search. See the VMS documentation set for details.) Each time the system translates a logical name, it checks to see if the result still contains a logical name. If so, the system translates the result. Therefore, you can define a logical name in terms of another logical name. You can also define more than one equivalence name for a single logical name. See the VMS documentation set for details about DCL commands.) For example:

```
$ ASSIGN "DBA2:[BELL.COURSE]"   HOMEWRK
$ ASSIGN "HOMEWRK:MONTHLY.TST"  TEST
```

In this example, the file specification DBA2:[BELL.COURSE] has been given the logical name HOMEWRK. This logical name is then used in the second file specification, HOMEWRK:MONTHLY.TST, to create the logical name TEST.

You can determine the current equivalence for a logical name by entering the SHOW LOGICAL command. For example:

```
$ SHOW LOGICAL HOMEWRK
HOMEWRK = "DBA2:[BELL.COURSES]"    (process)
```

To assign a logical name temporarily, use ASSIGN/USER_MODE command. For example:

```
$ ASSIGN/USER_MODE DMA0: DISK
```

The ASSIGN/USER_MODE command assigns the equivalence name DMA0: to the logical name DISK and stores the assignment in your process logical name table. The assignment is temporary and is deleted following completion of the next image. Because all but a few DCL commands complete an image when they execute, a user mode assignment of a logical name generally lasts only for one DCL command. Use this command to override existing logical names temporarily.

To cancel a logical name assignment, use the DEASSIGN command. For example:

```
$ DEASSIGN HOMEWRK
$ SHOW LOGICAL HOMEWRK
No translation for logical name HOMEWRK
```

There are many other attributes and options to control how the system interprets logical names. See the VMS documentation set for details.

2.11 Symbols

You can equate symbols to character strings or arithmetic values by defining them in assignment statements. In addition to their use in command procedures (see Section 2.12), symbols are useful as synonyms for long, frequently used command strings. For example, you can equate the symbol HOME to the command SET DEFAULT DBA2:[BELL] and subsequently use the symbol HOME in place of SET DEFAULT DBA2:[BELL]. For example:

```
$ HOME ::= SET DEFAULT DBA2:[BELL]
```

This symbol would be handy for bringing Bell back quickly to his main directory.

To display the current value of a symbol, use the SHOW SYMBOL command. For example:

```
$ SHOW SYMBOL HOME
HOME = "SET DEFAULT DBA2:[BELL]"
```

To delete a symbol, use the command `DELETE/SYMBOL/GLOBAL`. For example:

```
$ DELETE/SYMBOL/GLOBAL HOME
$ SHOW SYMBOL HOME
/DCL-W-UNDSYM, undefined symbol
```

The `DELETE/SYMBOL/GLOBAL` command deleted the symbol `HOME`. The response to the `SHOW SYMBOL` command verifies this.

2.12 Command Procedures

A command procedure is a file with the `.COM` type containing a sequence of commands to be executed by the operating system. You execute a command procedure with one command: the Execute Procedure character (`@`) for interactive processing or the `SUBMIT` command for batch processing.

As you continue to use `DCL`, you can simplify it to save yourself time during interactive terminal sessions and to establish your own default commands and command qualifiers.

You can use command procedures and symbol assignment statements together to redefine and expand system commands.

For example, suppose that during your terminal sessions you frequently create many files that you do not want cluttering up your directory. You may want to purge these files at the end of each session. To do this housekeeping, you could create a command procedure named `LOG.COM` that contains the lines:

```
$ PURGE
$ LOGOUT
```

You can use this command procedure in place of the `LOGOUT` command when you want to end your terminal session, as follows:

```
$ @LOG
```

The `PURGE` command is automatically executed before you log out.

Moreover, you could define a symbol named `LO` that is equated to the following command string:

```
$ LO ::= @LOG
```

Then, the system substitutes the symbol `LO` with the `@LOG` command string and executes your command procedure when you type the command line:

```
$ LO
```

2.12.1 A LOGIN.COM File

If you become a frequent user of the VMS system, you may find that you are entering the same sequence of commands or assignment statements every time you log in. To avoid such repetition, you can place these commands and statements in a special command procedure.

The command procedure file must be named LOGIN.COM, and it must be in your default disk directory. When you log in to the system, the system automatically searches for a file with this file name. If the system locates the LOGIN.COM file, it automatically executes the commands within that file.

For example, a LOGIN.COM file might contain:

```
$ ST ::= SHOW TIME
$ DIR*ECTORY ::= DIRECTORY/OWNER/PROTECTION
$ LD ::= @LOG
$ DTR32 ::= $SYS$SYSTEM:DTR32V3
$ SHOW PROCESS
```

Note that all the symbols defined in the previous example are global symbols, assigned with two equal signs. If these symbols were local (assigned with one equal sign), they would be recognized only within the LOGIN.COM file and would therefore be useless to you.

You can execute command procedures from within other command procedures. You may want to place the global assignment statements you use for command synonyms in a separate file and execute this procedure in the LOGIN.COM file. For example, suppose the file SYNONYM.COM contains the lines:

```
$ ST ::= SHOW TIME
$ DTR32 ::= $SYS$SYSTEM:DTR32V3
```

Your LOGIN.COM file would contain the line:

```
$ @SYNONYM
```

When this command is executed, the definitions in the synonym file are established.

2.13 Finding More Information

This chapter provides a brief overview of the VMS operating system. For more information, see the VMS documentation set.

Using Editors Within DATATRIEVE **3**

When you need to create or modify a dictionary object, you use an editor with its own set of rules, functions, commands and statements. This chapter discusses editing within DATATRIEVE. Within DATATRIEVE, you can use one of the following editors:

- EDT, which provides a basic editing interface and a predefined keypad with a variety of useful editing functions. EDT is the default editor within DATATRIEVE.
- VAX Text Processing Utility (VAXTPU), which allows multiple buffers and windows. VAXTPU allows you to tailor your editing interface to your individual editing style.
- VAX Language-Sensitive Editor (LSE), which has all the features of VAXTPU but also allows you to use DATATRIEVE LSE templates. These templates guide you to enter correct DATATRIEVE commands and statements.

Note that you can also edit from DCL level and at CDD level. At DCL level, you can use your choice of editors depending on what is installed on your system. At CDD level, you can use the Dictionary Management Utility (DMU) to edit CDD objects. See the documentation for your particular editor and for CDD for further information.

This chapter includes the following sections:

- General information you need to know to edit within DATATRIEVE
- Introductions to:
 - EDT
 - VAXTPU
 - LSE
- Information about ending your editing session

3.1 General Editing Information

This section discusses general information you need to know to edit within DATATRIEVE. It includes:

- Assigning a DATATRIEVE editor
- Using line recall within DATATRIEVE
- Using the DATATRIEVE EDIT command:
 - Editing the last DATATRIEVE command or statement
 - Using Access Control List privileges when editing CDD objects
 - Editing by CDD path name
 - Editing by types of objects within DATATRIEVE
 - Recovering an aborted editing session

3.1.1 Assigning a DATATRIEVE Editor

EDT is the default editor in DATATRIEVE. To change your default editor to VAXTPU or LSE, you need to change the editor assigned to the logical name DTR\$EDIT. You must use a three-character acronym, either EDT, TPU, or LSE, when you assign an editor to DTR\$EDIT.

You can assign an editor to DTR\$EDIT in one of two ways:

- Use the ASSIGN command at DCL level:

```
$ ASSIGN TPU DTR$EDIT
```

When you assign DTR\$EDIT with the DCL ASSIGN command, the assignment lasts only until you log out. After you log out, the previous default editor is again the default editor.

- Use the function FN\$CREATE_LOG from within DATATRIEVE:

```
DTR> FN$CREATE_LOG ("DTR$EDIT", "TPU")
```

When you assign DTR\$EDIT with FN\$CREATE_LOG, the assignment lasts only during that DATATRIEVE session. After you exit from DATATRIEVE, the previous default editor is again the default editor.

To assign an editor as your default editor whenever you use DATATRIEVE, include the ASSIGN command in your LOGIN.COM file. Your default DATATRIEVE editor will then be the editor assigned to DTR\$EDIT.

3.1.2 Using Line Recall Within DATATRIEVE

DATATRIEVE allows you to recall the 20 most recent input or prompted lines using the up arrow and down arrow keys or the CTRL/B key sequence.

You can use this feature to recall, correct, then reenter a previous line that contained an error. Line recall does not invoke an editor, however. To make your correction, recall the line with the error, position the cursor over the error using the arrow keys, then use the keyboard keys to insert or delete the necessary information.

- The up arrow key recalls lines in sequence from most recent to least recent.
- The CTRL/B key sequence also recalls lines in sequence from most recent to least recent.
- The down arrow key allows you to recall more recently entered lines after you have recalled prior lines.

You can use line recall at the DTR> prompt, the CON> prompt, and the RW> prompt. You can also recall prompted input lines, such as those generated by a prompting value expression (such as *.“prompt-name”) or a STORE or MODIFY statement. You cannot use line recall in Guide Mode, Help, or ADT.

Note that DATATRIEVE uses separate recall buffers for interactive and prompted input lines:

- If you are being prompted for input lines, you can recall the last 20 lines of prompted input.
- When the prompting has ended, DATATRIEVE switches to the interactive input buffer. Therefore, you can recall the previous 20 interactive input lines prior to the prompting session.
- If you subsequently enter another prompting session, you can recall the previous 20 lines from the previous prompting session.

In the following list, the position of the cursor is indicated by the underscore, . The list shows some unique situations using the recall feature:

- DATATRIEVE recalls lines continued with hyphens in interactive sessions as though they are one concatenated line.

The following example uses lines that have been continued in an interactive session with hyphens. The cursor is on the bottom line.

```
DTR> READY-  
CON> YACHTS-  
CON> SHARED-  
CON> READ  
DTR> _
```

If you press the up arrow key or CTRL/B once, DATATRIEVE recalls the command as one entire concatenated line:

```
DTR> READY YACHTS SHARED READ
```

Note that both interactive and prompted input lines have a maximum length of 255 characters.

- DATATRIEVE recalls a nonhyphenated, continued line in the same manner as it recalls a line entered at the DTR> prompt:

```
DTR> FIND YACHTS WITH BUILDER = "GRAMPIAN"  
DTR> PRINT ALL LOA,  
CON> BEAM,  
CON> DISP/2000 ("DISP/2000")  
DTR> _
```

As with a regular DTR> prompt, pressing the up arrow key or CTRL/B once recalls the most recent line, not the entire command or statement:

```
CON> DISP/2000 ("DISP/2000")
```

DATATRIEVE does not concatenate the recalled lines if they are not continued with hyphens.

- DATATRIEVE does not echo passwords on the screen, so you cannot edit or recall passwords.
- When DATATRIEVE prompts you for input, it interprets a hyphen as a minus sign, not as a continuation character. Thus, DATATRIEVE does not recall prompted lines as one continuous line if you have attempted to continue the lines with hyphens.

3.1.3 Using the DATATRIEVE EDIT command

The following sections discuss the DATATRIEVE EDIT command, including information on:

- Editing the previous command or statement
- Using Access Control List privileges when editing CDD objects
- Editing a CDD object specified by a path name
- Editing all objects of a particular type
- Recovering an aborted session

The EDIT section of the *VAX DATATRIEVE Reference Manual* also contains complete information on the EDIT command.

3.1.3.1 Editing the Previous DATATRIEVE Command or Statement — You can enter the EDIT command within DATATRIEVE without specifying a dictionary path name. DATATRIEVE then invokes your default editor and loads the previous command or statement into the main text buffer of the editor.

This feature is most useful if there was an error in the previous command or statement. The following list shows how you can use EDIT to correct such an error:

1. Enter the EDIT command with no argument.
2. DATATRIEVE loads the previous command or statement into the main text buffer.
3. Edit the previous command or statement to correct the error
4. Enter the EXIT command
5. DATATRIEVE executes the commands and statements that are in the editor's main buffer

In the following example, assume you did not want to include the argument BEAM:

```
DTR> FIND YACHTS WITH BUILDER = "GRAMPIAN"  
DTR> PRINT ALL LOA,  
CON> BEAM,  
CON> DISP/2000 ("DISP/2000")  
DTR>
```

To correct the mistake, type the EDIT command without an argument at the DTR> prompt; this recalls all the lines of the previous PRINT statement. Next, edit the statement, eliminating the BEAM argument. After you exit from the editor, DATATRIEVE executes the corrected statement:

```
DTR> PRINT ALL LOA,  
CON> DISP/2000 ("DISP/2000")
```

```
LENGTH  
OVER  
ALL      DISP/2000  
  
34      5.900  
26      2.800  
28      3.450  
30      4.300  
33      6.000
```

Note that you can use both the EDIT command and the arrow keys to recall and edit the previous line. The EDIT command and the arrow keys function differently, however, when you recall nonhyphenated commands or statements that are continued over more than one line. The following list shows these differences:

- The EDIT command recalls the *entire* last command or statement even if it spans more than one line.
- The arrow keys recall only a single line of a nonhyphenated, continued command or statement; they do not recall the entire command or statement. To correct an error in a nonhyphenated command or statement using the arrow keys, you would have to perform each of the following steps:
 1. Recall the first line of the command or statement and enter the RETURN key
 2. Recall each successive line of the command or statement and enter the RETURN key

3. When you reach the line where the error occurred, correct the error and enter the RETURN key
4. Recall any lines of the command or statement following the line where the error occurred, executing each line by entering the RETURN key

3.1.3.2 Privileges Needed to Edit CDD Objects — Each CDD object has an access control list (ACL) associated with it. ACLs determine what an individual user or class of users can do with an object. When you use an editor to define or redefine a dictionary definition within DATATRIEVE, the definition you are editing must have access privileges that allow you to create later versions. Typically, you need not worry about these privileges. The CDD is usually set up by the system manager to include the ACL privileges you need. See the chapter on using the VAX Common Data Dictionary for a description of privileges necessary to define and redefine definitions.

3.1.3.3 Editing a CDD Object Specified by Path Name — You can use EDT, VAXTPU, or LSE to create or modify existing CDD definitions. To create new CDD definitions, you can use the DEFINE command either within an editor or at the DTR> prompt. You can also use the Application Design Tool (ADT), which prompts you for information to create a new domain and record definition.

To edit a CDD object from within DATATRIEVE, enter the EDIT command followed by the CDD path name of the object:

```
DTR> EDIT definition-path-name
```

The editor then loads the specified definition into a **text buffer**, which is a temporary storage area where editing operations take place.

See the following sections on EDT, VAXTPU, and LSE for examples of editing a CDD object.

3.1.3.4 Editing by Types of Objects Within DATATRIEVE — You can specify one or more *types* of object definitions with the DATATRIEVE EDIT command. This allows you to edit *all* the domains, plots, procedures, records, or tables from your current default CDD directory.

DATATRIEVE places the object types in the edit buffer in the order you specify. You can then edit all the objects using EDT or your assigned editor. In the following example, DATATRIEVE places the record object definitions in the edit buffer before the domain object definitions:

```
DTR> EDIT ALL RECORDS, DOMAINS
```

See the EDIT command in the *VAX DATATRIEVE Reference Manual* for more information.

3.1.3.5 Using EDIT to Recover from a System Failure — Sometimes a computer system experiences problems that force it to shut down without warning. Your editor protects you from losing your editing work if this happens by creating a journal file that allows you to reconstruct your editing session.

While you are editing CDD objects, DATATRIEVE places a journal file for the editing session in your default VMS directory. The journal file is automatically deleted upon successful completion of the editing session. If your editing session ends abnormally, however, you can use the journal file and the RECOVER argument of the EDIT command to recover almost all the edits. The last several keystrokes may be missing.

To recover an aborted session, enter exactly the same line you entered when you started the session but add the RECOVER argument at the end of the line:

```
DTR> EDIT ALL DOMAINS
      .
      .! System failure
      .
DTR> EDIT ALL DOMAINS RECOVER
```

Journal files have default file types, depending on which editor you are using. You do not need to specify the journal file type when you are recovering an aborted session. You should know what the file type is, though, so you do not inadvertently delete the journal file before you recover the session. The following are the default file types:

EDT	.JOU
LSE	.TJL
VAXTPU	.TJL

If you are editing more than one type of object, DATATRIEVE creates a journal file using the name of the first object type:

```
DTR> EDIT ALL DOMAINS, RECORDS
```

In the preceding example, DATATRIEVE creates a journal file called DOMAINS.JOU.

Note that DATATRIEVE does not create a journal file when you are editing:

- The previous command or statement using the EDIT command with no arguments
- A previous line you have recalled using one of the arrow keys

Because there is no journal file, if your system fails during either of these situations, you cannot use the RECOVER argument.

3.2 Using EDT Within DATATRIEVE

To invoke EDT within DATATRIEVE, enter the EDIT command. As detailed in a preceding section, you can use several arguments with the EDIT command to achieve various results:

- To edit the previous command or statement, enter the EDIT command with no arguments.
- To edit a CDD object specified by path name, enter EDIT followed by a definition-path-name.
- To edit all objects of a particular type, enter EDIT ALL DOMAINS, EDIT ALL PLOTS, and so on.
- To recover an aborted session, enter the previous EDIT command followed by the argument RECOVER.

Although you can use any of the arguments mentioned in the previous list when you invoke EDT, the following example shows you how to use EDT to edit a CDD object specified by path name. Enter the EDIT command followed by a CDD definition path name:

```
DTR> EDIT definition-path-name
```

The editor then loads the specified definition into a text buffer, which is a temporary storage area where editing operations take place.

When you invoke EDT, the response varies depending on whether or not you are creating a new file or editing an existing file. Other factors, such as commands contained in an EDTINI.EDT startup command file, may further alter the response. See the EDT documentation for information on startup command files.

To edit the PHONES_REC record definition, type EDIT PHONES_REC at the DTR> prompt:

```
DTR> EDIT PHONES_REC
```

```
1          REDEFINE RECORD PHONES_REC USING
```

```
*
```

EDT puts a copy of the object definition in its buffer and then displays the first line of the definition and the asterisk (*) prompt. DATATRIEVE automatically adds the REDEFINE command to the beginning of the first line of PHONES_REC. The REDEFINE command creates a new version of the definition when you save any changes made during the editing session.

The asterisk is the EDT line editing prompt. When you see the asterisk prompt, it means you are in EDT line mode and not keypad mode. The following list describes EDT line mode and keypad mode:

- In line mode, you see only a single line of text at a time and perform editing operations on that line. You enter line editing commands by typing the command from the main keyboard.
- In keypad mode, you see an entire screen of text and can move freely about the display to edit at any point in the text. You enter keypad editing functions by pressing keys on the numeric keypad to the right of the main keyboard.

To use keypad mode, type the command CHANGE (or the abbreviation C) at the asterisk prompt. EDT switches to keypad mode and shows you a full screen of the record definition:

```
DTR> EDIT PHONES_REC
      1          REDEFINE RECORD PHONES_REC USING
*CHANGE
REDEFINE RECORD PHONES_REC USING
01 PHONES_REC.
   05 FULL_NAME      QUERY_NAME IS NAME.
   10 LAST_NAME     PIC X(20)
                   QUERY_NAME IS L.
   10 FIRST_NAME    PIC X(15)
                   QUERY_NAME IS F.
   05 PHONE_NUMBER  PIC X(8)
                   QUERY_NAME IS NUM.
;
[EOB]
```

The end-of-buffer symbol [EOB] indicates the last line of text.

See the section near the end of this chapter for information on leaving the editor.

3.3 Using VAXTPU Within DATATRIEVE

VAXTPU is a Text Processing Utility that allows you to tailor your editing environment to your own preferences. VAXTPU provides a variety of features not available with EDT, including:

- Multiple buffers, windows, and subprocesses
- The ability to define keys to execute a sequence of commands
- A procedural language

You have several choices of editing interfaces:

- The Extensible VAX Editor (EVE) has an easy-to-use editing interface that includes the most frequently used editing functions. You can also enter more advanced editing VAXTPU functions on an EVE command line and define your keyboard to suit your editing style. You can define keys to perform any VAXTPU/EVE commands or series of commands you choose. EVE is the default VAXTPU editing interface in DATATRIEVE.
- The EDT Keypad Emulator gives you an interface with the same editing keypad as the EDT editor. Like the EVE editing interface, it allows you to enter more advanced VAXTPU functions on an EDT Keypad Emulator command line and to define other keys on your keyboard to suit your editing style.
- If you prefer to design your own editing interface, you can define your keyboard the way you like it. You can define keys to perform any VAXTPU commands or series of commands or enter advanced VAXTPU functions on a command line.

Note

VAXTPU allows you to use a section file to modify your editing interface. When you use a section file, however, be careful not to specify, alter, or delete input and output file names. DATATRIEVE uses the file names DTR\$INPUT and DTR\$OUTPUT for VAXTPU. See VAXTPU documentation for more information about section files.

To use VAXTPU within DATATRIEVE, you must first assign VAXTPU as your default DATATRIEVE editor. (See the section on assigning a DATATRIEVE editor for more information.) This allows you to invoke the VAXTPU default editing interface, EVE.

If you want to invoke VAXTPU with the EDT Keypad Emulator rather than EVE, assign the logical name TPUSECINI. You can assign the logical name in one of two ways:

- Use the ASSIGN command at DCL level:

```
$ ASSIGN SYS$LIBRARY:EDTSECINI TPUSECINI
```

When you use the DCL ASSIGN command, the assignment lasts only until you log out. To assign the EDT Keypad Emulator as your default editor whenever you use DATATRIEVE, insert the ASSIGN command in your LOGIN.COM file.

- Use the function FN\$CREATE_LOG from within DATATRIEVE:

```
DTR> FN$CREATE_LOG ("TPUSECINI", "SYS$LIBRARY:EDTSECINI")
```

When you use FN\$CREATE_LOG, the assignment lasts only during that DATATRIEVE session.

To invoke VAXTPU within DATATRIEVE, enter the EDIT command. As detailed in a preceding section, you can use several arguments with the EDIT command to achieve various results:

- To edit the previous command or statement, enter the EDIT command with no arguments.
- To edit a CDD object specified by path name, enter EDIT followed by a path name.
- To edit all objects of a particular type, enter EDIT ALL DOMAINS, EDIT ALL PLOTS, and so on.
- To recover an aborted session, enter the previous EDIT command exactly as you entered it before, followed by the argument RECOVER.

While you can use any of the arguments from the previous list when invoking VAXTPU, the following example shows you how to use VAXTPU to edit a CDD object specified by path name. Invoke VAXTPU with the EDIT command followed by the CDD path name:

```
DTR> EDIT definition-path-name
```

VAXTPU then loads the specified definition into the editing buffer.

When you invoke VAXTPU, the response varies depending on whether or not you are creating a new file or editing an existing file. Other factors, such as commands contained in a startup command file, may further alter the response. See the VAXTPU documentation for information on startup command files.

To edit the PHONES_REC record definition, for example, type EDIT PHONES_REC at the DTR> prompt. VAXTPU puts a copy of the definition in the buffer and then displays the record definition:

```
DTR> EDIT PHONES_REC

REDEFINE RECORD PHONES_REC USING
01 PHONES_REC.
    05 FULL_NAME      QUERY_NAME IS NAME.
    10 LAST_NAME     PIC X(20)
                   QUERY_NAME IS L.
    10 FIRST_NAME    PIC X(15)
                   QUERY_NAME IS F.
    05 PHONE_NUMBER  PIC X(8)
                   QUERY_NAME IS NUM.
;

[End of file]
```

DATATRIEVE automatically adds the REDEFINE command to the beginning of the first line of PHONES_REC. The REDEFINE command creates a new version of the definition when you save any changes made during the editing session.

The [End of file] symbol indicates the last line of text in the buffer.

See the section near the end of this chapter for information on leaving the editor.

3.4 Using LSE within DATATRIEVE

LSE is based on VAXTPU and has all the editing features of VAXTPU.

LSE allows you to use a section file to modify your editing interface. When you use a section file, however, be careful not to specify, alter, or delete input and output file names. DATATRIEVE uses the file names DTR\$INPUT.DTR and DTR\$OUTPUT for LSE. See the LSE documentation for more information about section files.

In addition to the VAXTPU editing features, you can use DATATRIEVE LSE templates to guide you to enter correct commands and statements.

The DATATRIEVE LSE templates are made up of **placeholders**. The placeholders represent the DATATRIEVE syntax you need to define dictionary objects and to use DATATRIEVE. When you expand the placeholders, LSE provides the required syntax or indicates optional elements. You can expand these placeholders into:

- The required DATATRIEVE syntax elements that are appropriate for that context
- Optional elements

- Tokens representing appropriate keywords or information to be supplied
- Other placeholders

You expand a placeholder or token by positioning the cursor anywhere on it and pressing CTRL/E.

To use LSE within DATATRIEVE, you must first assign LSE as your default DATATRIEVE editor. (See the section in this chapter on assigning a DATATRIEVE editor for more information.)

Next, enter the EDIT command. As detailed in a preceding section, you can use several arguments with the EDIT command to achieve various results:

- To edit the previous command or statement, enter the EDIT command with no arguments.
- To edit a CDD object specified by path name, enter EDIT followed by a path name.
- To edit all objects of a particular type, enter EDIT ALL DOMAINS, EDIT ALL PLOTS, and so on.
- To recover an aborted session, enter the previous EDIT command followed by the argument RECOVER.

Although you can use any of the arguments in the previous list when invoking LSE, the following example shows you how to use LSE to edit a CDD object specified by path name. Invoke LSE with the EDIT command followed by the CDD path name:

```
DTR> EDIT definition-path-name
```

This command tells your editor to load the specified definition into the editing buffer.

When you invoke LSE, the response varies depending on whether or not you are creating a new file or editing an existing file. Other factors, such as commands contained in a startup command file, may further alter the response. See the VAX Language-Sensitive Editor documentation for further information on startup command files.

To edit the PHONES_REC record definition, for example, type EDIT PHONES_REC at the DTR> prompt. LSE puts a copy of the definition in its main buffer and then displays the record definition:

```
DTR> EDIT PHONES_REC
REDEFINE RECORD PHONES_REC USING
01 PHONES_REC.
    05 FULL_NAME          QUERY_NAME IS NAME.
      10 LAST_NAME       PIC X(20)
      QUERY_NAME IS L.
      10 FIRST_NAME     PIC X(15)
      QUERY_NAME IS F.
    05 PHONE_NUMBER     PIC X(8)
      QUERY_NAME IS NUM.
```

;

[End of file]

When the editor copies the record definition, DATATRIEVE automatically adds the REDEFINE command to the beginning of the first line of PHONES_REC. The REDEFINE command creates a new version of the definition when you exit the editor.

The [End of file] symbol indicates the last line of text in the buffer.

If you want some additional help in editing or want to use the general LSE template for DATATRIEVE, invoke the top-level placeholder in the following manner:

1. Invoke the editor.
2. Type the DATATRIEVE LSE placeholder {DATATRIEVE_session} at the top of the file.
3. Place your cursor on {DATATRIEVE_session}.
4. Enter CTRL/E to expand the placeholders and tokens included in the DATATRIEVE LSE templates.

To save keystrokes, you can define a key to print the placeholder {DATATRIEVE_session}. For example, adding this line to your LSE initialization file (LSE\$INITIALIZATION) enables the use of the key sequence GOLD-D for entering the placeholder into the edit buffer when you are in LSE:

```
DEFINE KEY/IF_STATE=GOLD D "DO ""ENTER TEXT {DATATRIEVE_session}"" "
```

See the following section for information on ending your editing session.

3.5 Ending Your Editing Session

Whether your default editor is EDT, VAXTPU, or LSE, you have two options when ending your editing session. You can:

- QUIT to end the session without preserving your work.
- EXIT to end the editing session but cause DATATRIEVE to execute any commands in the editing buffer. DATATRIEVE saves the original definition in the CDD when you exit the editor.

Note that when you exit, DATATRIEVE tries to execute the contents of your editing buffer. You should be sure your editing buffer contains valid DATATRIEVE syntax before you exit.

After you exit, a confirmation message and the DTR> prompt appear:

```
REDEFINE RECORD PHONES_REC USING
01 PHONES_REC.
    05 FULL_NAME      QUERY_NAME IS NAME.
    10 LAST_NAME     PIC X(20)
                   QUERY_NAME IS L.
    10 FIRST_NAME    PIC X(15)
                   QUERY_NAME IS F.
    05 PHONE_NUMBER  PIC X(8)
                   QUERY_NAME IS NUM.
;
[EOB]
CTRL/Z

*EXIT
[Record is 43 bytes long.]

DTR>
```

The following sections detail information specific to ending editing sessions from EDT, VAXTPU, and LSE.

3.5.1 Ending an EDT Session

To end an EDT editing session within DATATRIEVE, you must first leave EDT keypad mode by entering CTRL/Z. This returns you to line mode and the asterisk prompt.

At the asterisk prompt, end your editing session by typing either EXIT or QUIT, then press the RETURN key.

See the EDT documentation for more information about EDT commands and keypad editing.

3.5.2 Ending a VAXTPU Session

To EXIT from the EVE editing interface:

- On a VT100-family terminal, press CTRL/Z
- On a VT200-family terminal, press either the F10 key or CTRL/Z

To QUIT from the EVE editing interface:

- On a VT100-family terminal, press the PF4 key, then type QUIT and press the RETURN key
- On a 200-family terminal, press the DO key, then type QUIT and press the RETURN key

In the EDT Keypad Emulator, pressing CTRL/Z gives you an asterisk (*) prompt. You must then type either EXIT or QUIT and press the RETURN key to end your editing session.

See the VAXTPU documentation for more information about VAXTPU commands.

3.5.3 Ending an LSE Session

To end your LSE session, enter CTRL/Z. This moves your cursor down to the LSE> prompt at the bottom of the screen. You can then type either EXIT or QUIT, then press the RETURN key.

When you end your LSE editing session, do not leave any unexpanded placeholders in the file. Note that when you exit, DATATRIEVE tries to execute the contents of your editing buffer. DATATRIEVE can interpret only expanded placeholders and tokens; unexpanded placeholder and tokens will result in error messages.



Using Syntax Diagrams 4

This chapter explains the notation used in DATATRIEVE syntax diagrams.

The language you speak and write has a vocabulary and grammatical rules that make it unique and understandable to others. DATATRIEVE, too, contains a set of words and rules that determines how you can form statements and commands.

When you are first learning to use DATATRIEVE, you will probably use examples to guide you in writing your own statements, commands, and definitions. As you become more expert at using DATATRIEVE, however, you will find that no one example is complete enough for your particular application.

In the *VAX DATATRIEVE Reference Manual* and *VAX DATATRIEVE Pocket Guide* you will find a syntax diagram for each DATATRIEVE command, statement, and clause you can use. The syntax diagram tells you the words, order, and punctuation that apply to the language element illustrated. You can use syntax diagrams to supplement the information you gain from looking at examples.

DATATRIEVE online help also displays syntax diagrams for you. For example, to display the syntax diagram for the READY command, you can enter HELP READY at the DTR> prompt. Refer to Chapter 6 for more information about using the DATATRIEVE online help.

Figure 4-1 illustrates the syntax diagram for the READY command.

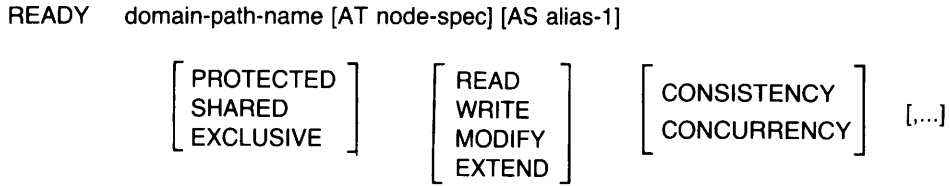


Figure 4-1: Sample Syntax Diagram

Table 4-1 explains the notation used in syntax diagrams.

Table 4-1: Notation Used in Syntax Diagrams

Element	Meaning	Do You Enter It?
WORD	An uppercase word is a keyword.	Yes
word	A lowercase word indicates a syntax element.	Yes
separators (punctuation)	Characters that separate words (space), separate items you are listing (comma), or tell DATATRIEVE you are finished with a clause (period) or a statement or command (semicolon).	Yes
{ } (braces)	Braces enclose a clause from which you must choose one alternative.	No
[] (brackets)	Square brackets enclose optional clauses from which you can choose or none.	No
... (horizontal ellipsis)	Indicates you can repeat the part of the clause, statement, command, or expression immediately to the left of the ellipsis	No
. (vertical) ellipsis)	Indicates you can repeat the line of the clause, statement, command, or expression immediately above the ellipsis	No

The format for the READY command indicates it always starts with the keyword READY and must contain at least one path name. The command READY EMPLOYEES, for example, contains the minimum number of elements that a READY command can include.

The following example illustrates a READY command that includes all the options in the diagram. It readies the domain FAMILIES on a local system for write access and specifies that other users cannot access the file. It also readies the domain PERSONNEL on a remote system for read access, specifies that other users can access the file, and specifies that the domain be readied under the name REM_PERSONNEL:

```
DTR> READY FAMILIES EXCLUSIVE WRITE, PERSONNEL AT
(Looking for Node Specification)
CON> WOMBAT"BELL *.'password'" AS REM_PERSONNEL SHARED READ
Enter password:
DTR> SHOW READY
Ready sources:
  REM_PERSONNEL: Domain, REMOTE, shared read
  FAMILIES: Domain, RMS sequential, exclusive write
  (CDD$TOP.DTR$USERS.BELL.FAMILIES)
No loaded tables.
```

The following examples illustrate what can happen when you input a statement or command that does not follow the rules in the syntax diagram. The comment line (DTR> !..) in each example indicates the correction needed. DATATRIEVE does not process comment lines. You generally add them only to procedures to document what is going on for someone reading the procedure. Sometimes in this book, comment lines are added to interactive examples to help you follow the steps being demonstrated:

```
DTR> READY FAMILIES WRITE AS FOLKS
READY FAMILIES SHARED WRITE AS FOLKS
      ^
Expected end of command, encountered "AS".
DTR> ! Alias clause should follow domain name
DTR> READY FAMILIES AS FOLKS WRITE
DTR>

DTR> READY FAMILIES FOLKS WRITE
READY FAMILIES FOLKS WRITE
      ^
Expected end of command, encountered "FOLKS".
DTR> ! Keyword AS is a required part of the alias clause
DTR> READY FAMILIES AS FOLKS WRITE
DTR>

DTR> READY FAMILIES READ PERSONNEL
READY FAMILIES READ PERSONNEL
      ^
Expected end of command, encountered "PERSONNEL".
DTR> ! Comma must separate information for the
DTR> ! two domains being readied
DTR> READY FAMILIES READ, PERSONNEL
DTR>
```


As you can probably guess, the error message “Expected end of command, encountered...” alerts you to a formatting problem with your input. When you receive this message, you should check the syntax diagram for the statement, command, clause, or expression you are trying to type.

The following sections discuss more fully the elements that make up syntax diagrams.

4.1 Uppercase Words in Syntax Diagrams

Uppercase words are DATATRIEVE keywords, the words that make up the main part of the DATATRIEVE language. Keywords can serve one of three purposes:

- “Major” keywords, such as READY, STORE, and FIND, let DATATRIEVE know what kind of task you want it to do. These keywords are always a required part of the input.
- “Minor” keywords, such as USING, AS, and ALL, sometimes let DATATRIEVE know what to expect next. In this case, these minor keywords are a required part of your input.
- “Minor” keywords, such as USING, AS, and ALL, can be included to make input more like English. In this case, the minor keyword is enclosed in square brackets and you can choose to omit the keywords from your input.


Do not use any DATATRIEVE keywords to name something you define. If you do, you might get either an error message or unexpected results. Appendix A contains a complete list of DATATRIEVE keywords.

4.2 Lowercase Words in Syntax Diagrams

Lowercase words indicate something that you supply to make a statement or command complete. The term domain-name limits you to path names of domains at the location where the term appears. The term statement means that you supply a complete DATATRIEVE statement in that location.

4.3 Brackets and Braces in Syntax Diagrams


Square brackets enclose optional parts of the language element being described by the syntax diagram. You can choose one or none of the optional clauses enclosed in square brackets. Braces enclose clauses from which you must choose one alternative. You can sometimes repeat the part of the diagram that is set off by square brackets or braces. The presence of the symbol [...] just outside the set tells you when brackets and braces are being used this way.



Generally, beginning users have difficulty reading brackets and braces correctly when these are nested inside one another. Try to remember that brackets and braces always travel in matched pairs. When you encounter a left bracket or left brace when reading a syntax diagram, immediately locate its mate on the right and look at what they enclose as a unit. As you are working through the first unit and encounter another bracket or brace, do the same thing again. Remember that when you decide to use a part of a diagram *within* a set of brackets or braces, each element inside is required unless it, in turn, is set off by a pair of brackets.

4.4 Separators in Syntax Diagrams

Separators let DATATRIEVE know that you are finished with something or, in the case of the comma, that you are not. Intuitively, you realize that typing a space after a word ends that word. One problem you might have with spaces is when you inadvertently put them in as replacements for underscores (__) or hyphens (-). For example, if you defined a domain with the name PHONE_LIST, DATATRIEVE would not know what you mean if you typed READY PHONE LIST in order to access the domain. DATATRIEVE would see three elements in the command where your more forgiving human eyes might see only two.



Get into the habit of typing a space after a comma. DATATRIEVE does not require that you do this, but if you do not, you might encounter a problem in some situations. For example, a comma preceding a string of editing characters could be interpreted as part of the string if there were no intervening space.

You must enter a comma to tell DATATRIEVE that you are entering more than one of the same kind of element. For example, commas are required to separate domain names in a READY or FINISH command or a list of things you want to display in a PRINT statement.

DATATRIEVE is more flexible about ending punctuation for statements and commands than are many other computer languages. It does require, however, that you type a period (.) to signal that you are finished describing a field (either a variable or a field in a record definition). It also requires that you type a semicolon (;) to signal that you are finished with a domain or record definition.



Part II
Controlling Your DATATRIEVE Environment





Input Options During a DATATRIEVE Session **5**

This chapter discusses the options you have when starting and ending a DATATRIEVE session and when entering input lines. The material in this chapter supplements the information on this topic in Chapter 1 and is not essential reading for DATATRIEVE beginners.

5.1 Invoking DATATRIEVE

When you invoke DATATRIEVE, you are running an image that resides in a system directory assigned to the logical name SYS\$SYSTEM. If you worked through the examples in Chapter 1, you created a logical symbol, DTR32, defined as \$SYS\$SYSTEM:DTR32 in your login command file. (Depending on how DATATRIEVE was installed on your system, the definition might have included a 2-character suffix on SYS\$SYSTEM:DTR32.)

After you create the symbol DTR32, you can use it in an invocation (foreign) command line, as well as to start an interactive DATATRIEVE session. In other words, you can use the symbol to execute one or more DATATRIEVE commands from a DCL command procedure and immediately return control to that command procedure. You must separate each of the DATATRIEVE commands or statements in a foreign command line with a semicolon.

The following example illustrates how the foreign command line operates. Note that you return to the dollar sign (\$) prompt after DATATRIEVE carries out your instructions:

```
$ DTR32 READY FAMILIES; PRINT FAMILIES WITH MOTHER = "ANN"
```

FATHER	MOTHER	NUMBER KIDS	KID NAME	AGE
JIM	ANN	2	URSULA	7
			RALPH	3

```
$
```

5.2 Creating a Startup Command File (DTR\$STARTUP)

If you frequently start your DATATRIEVE session with the same series of commands and statements, you can put them in a command procedure stored in one of your VMS directories. If you assign this file the logical name DTR\$STARTUP, its commands are executed every time you invoke DATATRIEVE.

The following is a sample of such a command file:

```
DECLARE SYNONYM SH FOR SHOW,  
          TY FOR PRINT,  
          PROC FOR PROCEDURES,  
          DEF FOR DICTIONARY,  
          DIRS FOR DICTIONARIES  
  
SH DEF  
SET NO PROMPT
```

The DECLARE SYNONYM allows you to create abbreviations or substitutions for DATATRIEVE keywords. After you create the synonym, you can use it in place of the keyword in your DATATRIEVE commands and statements. This can save you typing time and help you avoid confusion between DCL commands and DATATRIEVE equivalents. Be careful, however, that you do not create a synonym for a keyword that duplicates a different keyword or another synonym. You can check your synonyms against the list of keywords in Appendix A to ensure that each synonym is unique.

The SET NO PROMPT command is discussed later in this chapter.

Use EDT to create the startup file as you would any other file. Write the commands just as you would in an interactive session, but do not include any of the DATATRIEVE prompts.

Enter the following line in your LOGIN.COM file, substituting the file specification of your file for the one in the example:

```
$ ASSIGN "DBA1:[BELL]DTRSTART.COM" DTR$STARTUP
```

When you invoke DATATRIEVE, it translates the logical name DTR\$STARTUP and executes the command file before it displays the first DTR> prompt on your terminal. If you want your synonyms to take effect during the terminal session in which you define them, you can enter the command @LOGIN at the dollar sign (\$) prompt.

5.3 Exiting DATATRIEVE

You can exit DATATRIEVE in two ways:

- By typing EXIT at the DTR> prompt and pressing the RETURN key
- By entering CTRL/Z at the DTR> prompt

DATATRIEVE does not recognize the EXIT command when the CON>, DFN>, or RW> prompts are displayed. In this case, entering CTRL/Z returns you to the DTR> prompt and you can then exit your session. If you are in the middle of entering a statement or command and enter CTRL/Z, DATATRIEVE returns the message "Execution terminated by operator" to let you know that processing of your statement was not completed.

The following examples illustrate the two ways to end a DATATRIEVE session:

```
DTR> EXIT
$

DTR> READY
[Looking for dictionary path name]
CON> CTRL/Z
Execution terminated by operator.
DTR> CTRL/Z
$
```

Note

Entering CTRL/Y will also exit you from DATATRIEVE. CTRL/Y, however, also aborts any operations that you started and that are not yet complete. Using CTRL/Y to exit DATATRIEVE is not recommended.

5.4 Getting DATATRIEVE to Process More Than One Line as a Unit

Sometimes you need to enter statements or commands that are too long to fit on one line and must continue over two or more input lines. The easiest way to do this is to press the RETURN key after a word or character that is always followed by something else. Pressing the RETURN key following WITH, EQUALS, OF, or a comma, for example, lets DATATRIEVE know that you plan to continue. DATATRIEVE then displays a "Looking for..." message and a CON > prompt to tell you that it is waiting for more input.

If you press RETURN at the end of a line that is logically complete, no matter what your plans were for the next line, DATATRIEVE goes ahead and processes that line. The user entering the lines in the following example was not paying attention to the kind of prompt DATATRIEVE was giving her. A return to the DTR > prompt after pressing the RETURN key means that DATATRIEVE has processed the preceding input:

```
DTR> READY PERSONNEL
DTR> SHARED WRITE
SHARED WRITE
^
Expected statement, encountered "SHARED".
DTR> ! What is wrong?
DTR> SHOW READY
Ready sources:
  PERSONNEL: Domain, RMS indexed, protected read
              (CDD$TOP.DTR$USERS.BELL.PERSONNEL)
No loaded tables.

DTR> ! OK... adding SHARED should do it...
DTR> READY PERSONNEL SHARED
DTR> WRITE
WRITE
^
Expected statement, encountered "WRITE".
DTR> ! Now what is the problem...
DTR> SHOW READY
Ready sources:
  PERSONNEL: Domain, RMS indexed, shared read
              (CDD$TOP.DTR$USERS.BELL.PERSONNEL)
No loaded tables.

DTR> ! Sigh.... caught by a default....
DTR> READY PERSONNEL SHARED WRITE
DTR> SHOW READY
Ready sources:
  PERSONNEL: Domain, RMS indexed, shared write
              (CDD$TOP.DTR$USERS.BELL.PERSONNEL)
No loaded tables.

DTR> ! Got it....
```

If you were surprised to see that DATATRIEVE did not consider a line ending with SHARED incomplete, that is because READ access is a default access mode. DATATRIEVE assumes that you mean READ access if you do not specify an access mode. The line READY PERSONNEL SHARED, therefore, is a logically complete input line that DATATRIEVE can process.

5.4.1 Turning Off the "Looking for..." Messages

After you get the idea of where you can press the RETURN key to hold DATATRIEVE back from processing the input, you might find that the "Looking for..." messages needlessly clutter up your screen. If you want, you can turn these off (but still get a CON > prompt) by entering SET NO PROMPT:

```
DTR> READY FAMILIES
DTR> PRINT FAMILIES WITH
[Looking for Boolean expression]
CON> FATHER = "JIM" AND
[Looking for Boolean expression]
CON> MOTHER = "ANN"
```

FATHER	MOTHER	NUMBER KIDS	KID NAME	AGE
JIM	ANN	2	URSULA RALPH	8 4

```
DTR> SET NO PROMPT
DTR> PRINT FAMILIES WITH
CON> FATHER = "JIM" AND
CON> MOTHER = "ANN"
```

FATHER	MOTHER	NUMBER KIDS	KID NAME	AGE
JIM	ANN	2	URSULA RALPH	8 4

```
DTR>
```

5.4.2 Using Standard Programming Conventions to Format Input

This section is for programmers who have used continuation characters before and who prefer to keep words like WITH, OF, and EQUALS on the same line as the information that completes the phrases they begin.

DATATRIEVE uses the hyphen (-) as a continuation character. You can enter it at the end of a line to signal you are not finished with a command or statement. When you use the continuation character, you can break an input line where you want to. When the continuation character follows a word, however, you have to remember to enter a space character to separate that word from the next word. You can enter the space character before typing the hyphen or as the first character on the next line.

In the following example, RET indicates when the RETURN key is pressed:

```
DTR> PRINT FAMILIES WITH- (RET)
CON> FATHER = "JIM"- (RET)
CON> AND MOTHER = "ANN" (RET)
PRINT FAMILIES WITHFATHER = "JIM"AND MOTHER = "ANN"
```

Expected end of statement, encountered "WITHFATHER".

```
DTR> PRINT FAMILIES WITH - (RET)
CON> FATHER = "JIM"- (RET)
CON> AND MOTHER = "ANN" (RET)
```

FATHER	MOTHER	NUMBER KIDS	KID NAME	AGE
JIM	ANN	2	URSULA	8
			RALPH	4

DTR>

The SET SEMICOLON command is another option you can use to control input line format. When you enter SET SEMICOLON, DATATRIEVE considers any command or statement incomplete until you enter a semicolon (;). Semicolons can be a nuisance to remember. If you are already used to programming languages that require explicit statement termination, however, you might prefer this technique:

```
DTR> SET SEMICOLON
DTR> READY FAMILIES;
DTR> PRINT FAMILIES
CON> WITH FATHER = "JIM"
CON> AND MOTHER = "ANN"
CON> ;
```

FATHER	MOTHER	NUMBER KIDS	KID NAME	AGE
JIM	ANN	2	URSULA	8
			RALPH	4

DTR>

You can enter SET NO SEMICOLON to turn off the semicolon requirement.

Getting Online Assistance **6**

This chapter explains how to use the DATATRIEVE online Help facility, HELP ERROR, and Guide Mode.

6.1 Using Help

The DATATRIEVE HELP command provides online information about the use of DATATRIEVE commands, statements, and language elements.

When you enter HELP or a question mark (?) in response to the DTR> prompt, DATATRIEVE displays a list of topics to choose from.

If you already know which topic you want, you can enter it on the same line as the HELP command. For example, if you want information on defining a domain, enter:

```
DTR> HELP DEFINE DOMAIN
```

You can do this with any topic available in help. ERROR is one of these topics, but HELP ERROR follows somewhat different rules. See the next section for information on how to use HELP ERROR.

When you are in the Help facility, press the PF2 key on the auxiliary keypad or enter VIDEO as the topic. This displays information on the screen-oriented Help facility and explains how to scan the DATATRIEVE help messages. You can move through the text by using the arrow keys. Press the:

- Up arrow to scroll the help text backward to previous lines
- Down arrow to scroll the help text forward
- Left arrow to display the previous complete help screen
- Right arrow to display the next complete help screen

You can type a question mark (?) to display the current help topics again.

If you enter HELP HELP, DATATRIEVE displays more detailed information on the HELP command.

If there are any subtopics of the topic you have selected, DATATRIEVE prompts you to choose for additional information.

If you are at one of the subtopic levels of help, you can press the RETURN key to move up a level. This is necessary if you want to select a topic displayed on a previous level. The prompt displayed tells you at what level of help you are. For example, the prompt "DEFINE DOMAIN subtopic?" tells you that you are two levels down in help. If you press the RETURN key, you see the prompt "DEFINE subtopic?" and can enter another DEFINE selection, such as RECORD.

Enter CTRL/Z to exit from help. This returns you to the DATATRIEVE prompt DTR>.

6.2 Getting Help on Errors

When DATATRIEVE displays an error message, you can type HELP ERROR and DATATRIEVE displays the help text pertaining to that error. For example:

```
DTR> FIND PERSONNEL
"PERSONNEL" is not a readied source, collection, or list.
DTR> HELP ERROR
"PERSONNEL" is not a readied source, collection, or list.
```

ERROR

NOTDOMAIN

EXPLANATION:

The source for a DATATRIEVE collection must be a readied domain, relation, or DBMS record; a collection; or a list.

USER ACTION:

Check that you have spelled all names correctly. Ready the appropriate record source, if necessary, and reenter the statement.

Topic? **CTRLZ**

DTR>

Note

DATATRIEVE always gives you information on the last error you made, even if it was many commands ago.

If you have not made any error during a DATATRIEVE session, entering **HELP ERROR** gives you a display of all the error topics. To get the same display after you have made an error, you may enter **ERROR** when you are at the Topic? prompt in help.

6.3 Guide Mode

Guide Mode is a self-documenting aid available whenever you are at the DTR> prompt. To enter Guide Mode, type:

DTR> **SET GUIDE**

If you are using anything but a VT100, VT200, or compatible terminal, DATATRIEVE displays an error message that tells you your terminal type is invalid. DATATRIEVE then returns you to the DTR> prompt.

Guide Mode has two functions:

- To complete typing an entry for you
- To prompt you for a legitimate entry

As you enter each word of a command or statement, you can still enter the word as you usually do. As soon as you have typed enough letters to identify only one possible choice, however, you can press the space bar and Guide Mode completes the entry for you and prompts you for the next word. At the end of a line, press the RETURN key and Guide Mode goes to the next line.

When Guide Mode is waiting for your input, you can press the question mark (?) key. Guide Mode then displays a list of all the words you can use at that point.

DATATRIEVE also supplies you with Advanced Guide Mode. To enter this type of Guide Mode, enter:

```
DTR> SET GUIDE ADVANCED
```

This functions exactly like regular Guide Mode, except that more words are usually available as prompts. The choice of words is made when DATATRIEVE is installed. By default, the PLOT and REPORT statements, and the use of a colon (:) to invoke a procedure are available in advanced Guide Mode only. The following words are available at both levels by default:

FIND	MODIFY	PRINT
READY	SELECT	SET
SHOW	SORT	STORE

The easiest way to learn about Guide Mode is to use it. You may find it particularly helpful when you are starting to use DATATRIEVE. Experiment with it and use it the way it helps you the most.

Using the VAX Common Data Dictionary **7**

This chapter gives you basic information about the VAX Common Data Dictionary (CDD) so that you understand its relation to DATATRIEVE. (VAX CDD software is also referred to throughout this manual simply as CDD.) It also explains how to create dictionary directories, display information about directories and their contents, determine access privileges for directories and definitions, and delete definitions and directories.

The information in this chapter supplements the information on the CDD in Chapter 1. The details about CDD structure and access privileges are important to users who are creating DATATRIEVE applications that will be used by people other than themselves. If you are not in that category and are just beginning to use DATATRIEVE, you might want to skip this chapter for now and return to it when you have questions about specific CDD topics.

7.1 What Is the CDD?

The CDD is a central repository for data definitions. It can:

- Store data definitions such as DATATRIEVE record, domain, view, and table definitions
- Store DATATRIEVE procedure definitions
- Keep information about the location of each definition
- Control the access to each definition
- Keep track of what happens to each definition—when and by whom a definition is changed and how each definition is being used

The CDD can be used by traditional programming languages such as BASIC and COBOL, as well as by DATATRIEVE. It solves the problems of data redundancy and inconsistency by keeping central record definitions that a variety of languages can use. Data is therefore no longer tied to a particular program, and programs written in a variety of languages can access the same data file.

7.2 How Is the CDD Organized?

The CDD is organized as a hierarchy of dictionary directories and dictionary objects. Dictionary directories are similar to VMS directories in that they organize information within the hierarchy. Data definitions are dictionary objects. The definitions are contained in the directories just as files are contained in VMS directories, and they are located at the ends of the branches in the hierarchy.

The CDD hierarchical structure is like a family tree. Dictionary directories are the parents, and their children include other directories, as well as dictionary objects. Figure 7-1 illustrates the tree structure of the CDD.

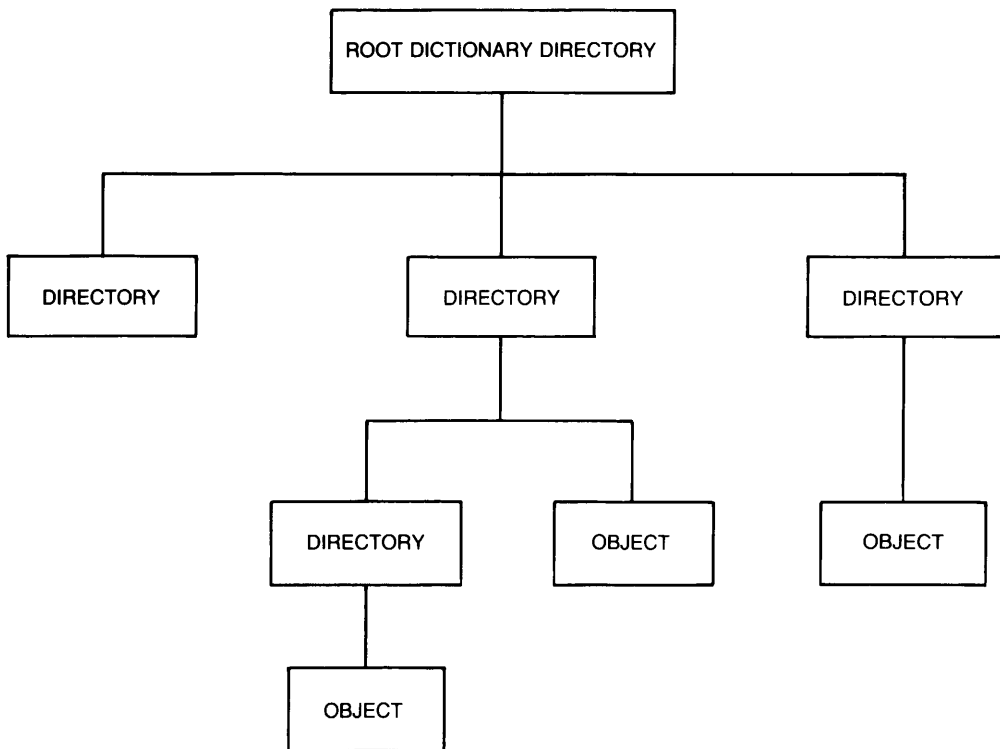
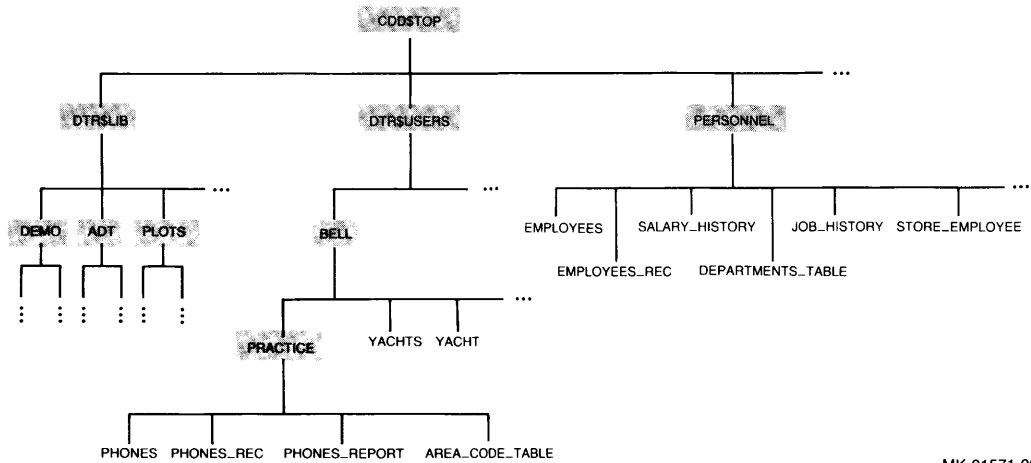


Figure 7-1: CDD Structure

MK-00680-00

Figure 7-2 illustrates a sample CDD. Shaded boxes indicate directories and unshaded names indicate objects. An ellipsis (...) indicates that the CDD branching continues, but is not shown in the figure. (Few CDD directories are small and symmetrical enough to fit neatly on one page of a book!) The examples in this book are drawn from this sample dictionary and its associated data definitions.



MK-01571-00

Figure 7-2: Sample CDD

You can see that all directories and objects are descendants of CDD\$TOP. CDD\$TOP is found at the top of every CDD and is created when the CDD is installed.

DTR\$USERS is a directory under CDD\$TOP that can be created during DATATRIEVE installation as a parent directory for the private directories of DATATRIEVE users. BELL is a directory created by the DATATRIEVE NEWUSER program. It contains the sample definitions copied into it by the NEWUSER program, as well as the PRACTICE directory created by user Bell to store the definitions PHONES and PHONES_REC.

DTR\$LIB is a directory under CDD\$TOP that, along with its subdirectories and the definitions they contain, is always created by the DATATRIEVE installation procedure. Later on, you might want to use the SET DICTIONARY and SHOW commands to become familiar with what the DTR\$LIB branch of the CDD contains. You should never create a dictionary or store your own definitions anywhere in the DTR\$LIB branch of the CDD. DTR\$LIB and all of its descendants are deleted and rebuilt each time a new version of DATATRIEVE is installed on your system.

PERSONNEL is a directory under CDD\$TOP that contains the definitions for the personnel system examples used in later chapters of this book.

7.3 Creating and Using Path Names

Every directory and object in the CDD has a given name, assigned by the installation procedure or person who created it. Two objects in the CDD can have the same given name, but they must reside in different directories.

The CDD has rules to which names must conform. Given names must:

- Begin with a letter (A-Z)
- End with a letter or a digit (A-Z, 0-9)
- Not exceed 31 characters in length
- Contain only letters, digits, dollar signs, underscores, or hyphens

You can also use 8-bit alphabetic characters in CDD path names. As far as CDD names are concerned, DATATRIEVE converts lowercase letters in names to uppercase. DATATRIEVE also treats an underscore and a hyphen (-) as the same character. If you type phones-table for the name of an object, for example, DATATRIEVE interprets the string as PHONES_TABLE and that is how you see the name displayed.

Note

As far as field *values* are concerned, however, DATATRIEVE does not perform case conversion. If you store an employee's name using mixed case or a hyphen (Smith-Donlevey, for example), DATATRIEVE stores it exactly as you typed it. If you later try to find the name by typing all uppercase or lowercase letters or substitute an underscore for a hyphen (SMITH_DONLEVEY, for example), DATATRIEVE will not recognize it. It is important, therefore, to understand the distinction between 1) the names you use to identify directories, objects, and record fields and 2) the values stored in data files.

If you define or assign a logical name either inside or outside DATATRIEVE that duplicates the name of a CDD object, you may get unexpected results.

For example, you can run into problems if you assign a logical name for YACHTS. In the following case, the user assigned the logical name in DATATRIEVE with the FN\$CREATE_LOG function:

```
DTR> FN$CREATE_LOG ("YACHTS", "DB2:[MORRISON]LOGIN.COM")
DTR> PRINT FN$TRANS_LOG ("YACHTS")
```

```
FN$TRANS
LOG
```

```
DB2:[MORRISON]LOGIN.COM
```

```
DTR> READY YACHTS
XCDD-E-ILLNAMCHR, A given name contains a character other than A-Z, 0-9, $, _
```

DATATRIEVE could not ready the domain because CDD had translated the logical name "YACHTS" into a file specification. If you begin the domain name with an underscore, you avoid the problem:

```
DTR> READY _YACHTS
DTR>
```

Another solution is to delete the logical name assignment within DATATRIEVE by using FN\$DELETE_LOG. Now CDD understands YACHTS to be the name of a CDD object in the default dictionary. CDD can access YACHTS, and DATATRIEVE can ready the domain:

```
DTR> FINISH
DTR> FN$DELETE_LOG ("YACHTS")
DTR> READY YACHTS
DTR> SHOW READY
Ready sources:
  YACHTS: Domain, RMS indexed, protected read
         (CDD$TOP.DTR32.MORRIS.YACHTS)
No loaded tables.
```

7.3.1 Using Full Path Names

You have to travel down a path from CDD\$TOP to reach a target directory or object. You specify the path name of a directory or object by linking together the names of all the directories starting with CDD\$TOP and ending with the given name of the target directory or object. Each name in the path name is separated from the others by a period. The full path name of a dictionary object also includes a semicolon (;) followed by a version number:

```
DTR> SHOW DOMAINS
Domains:
  COLLEGES;1      DEGREES;1      EMPLOYEES;1    JOBS;1
  JOB_HISTORY;3  JOB_HISTORY;2  JOB_HISTORY;1
  SALARY_HISTORY;1
```

```
DTR>
```

The full path name of the most recent version of the JOB_HISTORY domain in the preceding display is CDD\$TOP.PERSONNEL.JOB_HISTORY;3.

Versions of dictionary objects serve the same purpose as versions of VMS files: they provide backup security to protect you when you change their contents.

7.3.2 Version Numbers

Most CDD objects have a version number. The CDD can contain more than a single version of all CDD objects except dictionaries (domains, procedures, records, and so on). For the most part, the rules for specifying versions of DATATRIEVE definitions follow the rules for specifying versions of VMS files.

The version number is separated from the rest of the given name by a semicolon. This number can be an absolute version number or a relative version number. You can omit the semicolon and number but if you use the semicolon without a number, DATATRIEVE considers the semicolon the end of a command or statement.

Table 7-1 lists the various ways you can specify version numbers, the result of each specific way, and an example of each way.

Table 7-1: Specifying Version Numbers

Specification	Result	Example
Absolute version number	DTR operates on the object with the specified version number.	SALARY_RANGE;2
Relative version number*	DTR operates on the object at a specified number below the highest version.	SALARY_RANGE;-1
No semicolon or version number	DTR operates on the highest version of the object.	SALARY_RANGE

* You cannot use this specification with the REDEFINE or DEFINE command.

For more information on CDD version support, see the *VAX Common Data Dictionary* documentation.

7.3.3 Abbreviating Path Names

You do not always need to use the full dictionary path name to identify directories and objects in the CDD. Your abbreviation depends on where the target directory or object is, *relative* to your current position in the CDD. 'Relative path name', in fact, is the proper term for a path name abbreviation.

If you are looking downward in the CDD tree structure (away from CDD\$TOP), you have to specify only the portion of the path name below the level of your current dictionary location. If your dictionary location is at CDD\$TOP.DTR\$USERS.BELL in the sample dictionary, for example, DATATRIEVE will understand the following commands:

```
SHOW PRACTICE.PHONES
SHOW PRACTICE.PHONES;
SHOW PRACTICE.PHONES;2
SET DICTIONARY PRACTICE
```

If you type a semicolon without a version number, or simply omit it altogether, DATATRIEVE uses the PHONES definition with the highest version number.

If you have to "back up" toward CDD\$TOP to get to your target directory or object, you can substitute a hyphen (-) in your path name for each directory name leading to CDD\$TOP until you have entered one for the first dictionary directory common to both your current location and the path name you want to specify. A later section in this chapter contains examples of using hyphens in path names.

7.3.4 The Logical Name in Dictionary Path Names

You can use logical names to refer to other objects, as well as to give you an easy means for specifying long dictionary path names.

For example, you have a DCL command in your LOGIN.COM file that defines CDD\$DEFAULT as the logical name for CDD\$TOP.DTR\$USERS.BELL. However, you occasionally work with the sample definitions and dictionaries in the CDD\$TOP.DTR\$LIB.DEMO directory. Rather than enter the full path name of the DEMO directory, you can use a logical name when you change directories. You can put this DCL command in your LOGIN.COM file:

```
$ DEFINE SAMPLE "CDD$TOP.DTR$LIB.DEMO"
```

Then, when you want to change to your default directory, enter this DATATRIEVE SET command:

```
DTR> SET DICTIONARY SAMPLE
DTR> SHOW DICTIONARY
The default directory is CDD$TOP.DTR$LIB.DEMO
```

```
DTR>
```

To change back to CDD\$TOP.DTR\$USERS.BELL, enter this command:

```
DTR> SET DICTIONARY CDD$DEFAULT
DTR> SHOW DICTIONARY
The default directory is CDD$TOP.DTR$USERS.BELL
```

```
DTR>
```

You can form valid dictionary path names by combining logical names with the names of directories and objects. You must put the logical name first, followed by the given names. For example, your default directory is CDD\$TOP.DTR\$USERS.BELL. You want to ready the FAMILIES domain cataloged in the DEMO directory, but you do not want to change default directories. You can enter this READY command:

```
DTR> READY SAMPLE.FAMILIES
DTR> SHOW READY
Ready sources:

    FAMILIES: Domain, RMS indexed, protected read
              <CDD$TOP.DTR$LIB.DEMO.FAMILIES;1>
No loaded tables.
```

```
DTR> SHOW DICTIONARY
The default directory is CDD$TOP.DTR$USERS.BELL
```

```
DTR>
```

Note

Do not define your own logical names beginning with three letters and a dollar sign (\$). You especially must avoid defining your own logical names beginning with DTR\$, which is reserved for use by DATATRIEVE.

7.4 Setting Dictionary Location

When you invoke DATATRIEVE, your location in the dictionary is the dictionary directory assigned to the logical name CDD\$DEFAULT. If you do not have an assignment for CDD\$DEFAULT and invoke DATATRIEVE, your dictionary location is CDD\$TOP.

Use the DATATRIEVE SET DICTIONARY command to move to another directory. You can include either a full or relative path name for your destination. You can also use the logical name CDD\$DEFAULT to return to the directory you assigned to it. Because a hyphen at the end of an input line is always interpreted as a continuation character, put a semicolon (;) at the end of any SET commands that end in a hyphen:

```
DTR> SET DICTIONARY CDD$DEFAULT
DTR> SHOW DICTIONARY
The default dictionary is CDD$TOP.DTR$USERS.BELL

DTR> SET DICTIONARY CDD$TOP.DTR$LIB.DEMO
DTR> SHOW DICTIONARY
The default dictionary is CDD$TOP.DTR$LIB.DEMO

DTR> SET DICTIONARY -.-.DTR$USERS.BELL
DTR> SHOW DICTIONARY
The default dictionary is CDD$TOP.DTR$USERS.BELL

DTR> SET DICTIONARY PRACTICE
DTR> SHOW DICTIONARY
The default dictionary is CDD$TOP.DTR$USERS.BELL.PRACTICE

DTR> SET DICTIONARY -;
DTR> SHOW DICTIONARY
The default dictionary is CDD$TOP.DTR$USERS.BELL

DTR>
```

7.5 Displaying Information About Directories, Objects, and Session Defaults

Use the SHOW command to display information stored in dictionary directories. (The DATATRIEVE PRINT and LIST statements, on the other hand, display *data* stored in VMS directories.) Table 7-2 describes most of the options you have with the SHOW command. The table does not include information about special help display formats, data managed by VAX DBMS or relational products, or display by forms.

Table 7-2: SHOW Command Options

Option	Result
SHOW ALL	Displays the names of all the objects and directories cataloged in your default directory, the name of your default directory, and the names of the collections, the readied domains, and the loaded tables you are using.
SHOW collection-name	Displays the collection name, the name of the domain within which the collection has been established, the number of records in the collection, the status of the selected record within the collection, and the names of the keys on which the collection has been sorted.
SHOW COLLECTIONS	Displays the names of the collections you are using.
SHOW CURRENT	Displays the name of the domain within which the CURRENT collection has been formed, the number of records in the CURRENT collection, the status of the selected record in the CURRENT collection, and the names of the keys on which the collection has been sorted.
SHOW DICTIONARIES	Displays the names of the CDD directories appended to your default directory. (This option tells you if the CDD branch continues lower than your current location.)
SHOW DICTIONARY	Displays the full dictionary path name of your default directory. (This option answers the question "Where am I?".)
SHOW DOMAINS	Displays the names of all domains cataloged in your default directory.
SHOW EDIT	Indicates whether SET EDIT_BACKUP or SET NO EDIT_BACKUP is in effect in your DATATRIEVE session. SET EDIT_BACKUP is the default. If you enter the command SET NO EDIT_BACKUP and edit any definitions, DATATRIEVE deletes the highest version of the definitions when you exit the editor. SET NO EDIT_BACKUP automatically keeps outdated versions of definitions from piling up in your directory, but could erase the only definition you have to fall back on should you make a mistake in your editing.
SHOW FIELDS	Displays the names, data types, and index-key information of the fields of all domains you have readied. It also displays the names and data types of global variables.

(continued on next page)

Table 7-2: SHOW Command Options (Cont.)

Option	Result
SHOW FIELDS FOR domain-name	Displays the names, data types, and FOR domain-name index-key information of the fields in the domain you specify after FOR. You can only specify the name of a readied domain.
SHOW KEYDEFS	Shows all current key definitions in all states. A state allows the same key to be assigned multiple definitions by associating each definition with a different state key.
SHOW path-name	Displays the text of the domain, record, procedure, or table definition you specify.
SHOW PRIVILEGES	Displays the access privileges you have to the directory at which you are currently located.
SHOW PRIVILEGES FOR path-name	Displays the access privileges you have to the directory or object you name in the FOR clause.
SHOW PROCEDURES	Displays the names of all procedures cataloged in the directory at which you are currently located.
SHOW READY	Displays for each readied domain the full dictionary path name, the file organization of the associated data file, the access control option (EXCLUSIVE, PROTECTED, or SHARED), and the access mode (READ, WRITE, EXTEND, or MODIFY). The most recently readied domain is at the top of the list displayed on your terminal. The SHOW READY command also displays the full dictionary path name and table type of all tables you have loaded into your work area.
SHOW RECORDS	Displays the names of all record definitions cataloged at your current directory location.
SHOW SET_UP	Displays the current status of the options you can control with the SET command: ABORT/NO ABORT, APPLICATION_KEYPAD/ NO APPLICATION_KEYPAD, COLUMNS_PAGE, FORM/NO FORM, PROMPT/NO PROMPT, SEARCH/NO SEARCH, SEMICOLON/ NO SEMICOLON, and VERIFY/NOVERIFY.

(continued on next page)

Table 7-2: SHOW Command Options (Cont.)

Option	Result
SHOW SYNONYMS	Displays the names of any synonyms for DATATRIEVE keywords in effect during your DATATRIEVE session.
SHOW TABLES	Displays the names of all dictionary tables and domain tables cataloged in your default directory.
SHOW VARIABLES	Displays the global variables in effect in the current DATATRIEVE session.

There is more information on SHOW PRIVILEGES in a later section of this chapter. Variables and collections are concepts discussed in later chapters.

7.6 Deleting, Purging, and Extracting Definitions

Use the DELETE command to erase definitions from dictionary directories. When you delete a definition, you must always include an explicit version number and a semicolon to end the command. This means that every DELETE command contains at least two semicolons:

```
DTR> SHOW RECORDS
Records:
      PHONES_REC;3    PHONES_REC;2    PHONES_REC;1

DTR> DELETE PHONES_REC;
%CDD-E-VERNUMREQ, version number required on object name
DTR> DELETE PHONES_REC;1
DELETE PHONES_REC;1
      ^

Expected end of command, encountered "***END_OF_LINE***".
DTR> DELETE PHONES_REC;1;
DTR> SHOW RECORDS
Records:
      PHONES_REC;3    PHONES_REC;2

DTR>
```

You can get rid of outdated versions of definitions using one of two methods:

- You can explicitly delete each version of the definition you do not want to keep.
- You can use the PURGE command to delete all but the highest version or specified versions of the definition.

To purge CDD objects, set your dictionary location to the directory containing the definitions you want to purge. Enter the PURGE command or the PURGE command with the KEEP argument to delete outdated versions of definitions.

The following example shows how user Bell purges all but the two highest versions of PHONES_REC in his PRACTICE directory:

```
DTR> SHOW DICTIONARY
The default directory is CDD$TOP.DTR$USERS.BELL

DTR> SET DICTIONARY CDD$TOP.DTR$USERS.BELL.PRACTICE
DTR> SHOW RECORDS
Records:
    PHONES_REC;4    PHONES_REC;3    PHONES_REC;2

DTR> PURGE PHONES_REC KEEP = 2
DTR> SHOW RECORDS
Records:
    PHONES_REC;4    PHONES_REC;3
```

If you want to move a definition to another dictionary directory or send it to another user on your system, you can use the EXTRACT command to copy the definition into a VMS file that you can execute or send. You (or the other user) can then use the at sign (@) to store the definition at a new dictionary location.

The following example shows how user Bell copies the definition SALES_REC in the directory CDD\$TOP.DTR\$LIB.DEMO to the file TEMP.COM, sets his dictionary location to his own directory, and stores the SALES_REC definition in his own dictionary directory. Note that before he executes the file, Bell checks to be sure there is nothing in his own directory with the same name as the definition in TEMP.COM. If there were, he would edit one of the definitions to change the object name:

```
DTR> SET DICTIONARY CDD$TOP.DTR$LIB.DEMO
DTR> SHOW RECORDS
Records:
    ANNUAL_REC;1    DAB;1            FAMILY_REC;1    OWNER_RECORD;1
    PAYABLES_REC;1 PERSONNEL_REC;1  PET_REC;1      PROJECT_REC;1
    SALES_REC;1    YACHT;1

DTR> EXTRACT SALES_REC ON TEMP.COM
DTR> SET DICTIONARY CDD$TOP.DTR$USERS.BELL
DTR> SHOW ALL
```

(continued on next page)

```
Domains:
  FAMILIES;1    OWNERS;1    PERSONNEL;1    PETS;1
  PROJECTS;1   YACHTS;1

Records:
  FAMILY_REC;1  OWNER_RECORD;1  PERSONNEL_REC;1  PET_REC;1
  PROJECT_REC;1  YACHT;1

The default directory is CDD$TOP.DTR$USERS.BELL
No established collections.
```

No ready sources.

No loaded tables.

```
DTR> @TEMP.COM
```

Element "SALES_REC" not found in dictionary.

[Record is 35 bytes long.]

Element to be redefined not found in dictionary - new element defined.

```
DTR> SHOW RECORDS
```

```
Records:
  FAMILY_REC;1    OWNER_RECORD;1  PERSONNEL_REC;1  PET_REC;1
  PROJECT_REC;1  SALES_REC;1    YACHT;1
```

```
DTR>
```

The messages resulting from the store operation at the new location are informational and do not indicate a problem. The extract operation automatically puts DELETE and REDEFINE commands before the definition in TEMP.COM. In Bell's directory, nothing can be deleted or redefined as a new version, so DATATRIEVE lets him know that. The REDEFINE command still stores the definition for him.

7.7 Creating Dictionary Directories

You can append new directories to your branch of the CDD with the DEFINE DICTIONARY command.

Note

Depending on the privileges you have, you might find that you can create directories in other branches of the CDD. Check with your CDD manager before you do this, however. For best CDD management, users on a system should coordinate where they create directories and store definitions. In addition, some branches of the CDD, especially those created by DIGITAL products, are periodically deleted and rebuilt. If you store definitions in these branches, you could eventually lose them. The DATATRIEVE installation creates the DTR\$LIB branch of the CDD, for example, and you should not store definitions there.

If you specify only the given name of the new directory, DATATRIEVE appends the directory to the one at which you are currently located:

```
DTR> SHOW DICTIONARY
The default directory is CDD$TOP.DTR$USERS.BELL

DTR> DEFINE DICTIONARY SALES
DTR> SHOW DICTIONARIES
Dictionaries:
      PRACTICE      SALES

DTR>
```

7.8 Deleting Dictionary Directories

You cannot delete any of your directories from DATATRIEVE command level. You must exit DATATRIEVE, invoke DMU, and use the DMU DELETE command to do this. The following example illustrates the procedure used by Bell to delete his PRACTICE directory:

```
$ RUN SYS$SYSTEM:DMU
DMU> SHOW DEFAULT
CDD$TOP.DTR$USERS.BELL
DMU> SET DEFAULT PRACTICE
DMU> LIST
  AREA_CODE_TAB;1 <CDD$TABLE>
  PHONES;1 <DTR$DOMAIN>
  PHONES_REC;4 <CDD$RECORD>
  PHONES_REPORT;9 <DTR$PROCEDURE>
DMU> DELETE *;*
DMU> LIST
ZDMU-E-NONODFND, no directories or objects found
DMU> SET DEFAULT CDD$TOP.DTR$USERS.BELL
DMU> LIST/TYPE=DIRECTORY
  PRACTICE
  SALES
DMU> DELETE PRACTICE
DMU> LIST/TYPE=DIRECTORY
  SALES
DMU> EXIT
$
```

Note that he had to empty PRACTICE of all its contents before deleting the directory itself. If he had appended subdirectories to PRACTICE, he would have had to start the delete procedure at the lowest level of his branch in the dictionary and delete his way up each twig before he could delete PRACTICE.

DMU also has a DELETE/ALL command that can wipe out an entire branch of the CDD in one line of input. A user needs a special CDD privilege (GLOBAL_DELETE) to be able to use this command. The average CDD user, understandably, does not get this privilege by default and usually is not assigned it by the person who manages the CDD. The following section discusses CDD access privileges more fully.

7.9 Displaying and Setting Protection for Directories and Objects

The key to the CDD system of protection is the access control list (ACL). Each dictionary directory and object has an ACL associated with it. ACLs determine whether an individual user or class of users can:

- Create, modify, or delete a dictionary directory or object
- See the definition of an object
- Use the object definition in an application and, if so, for what kind of operation
- See or modify the information in the history list associated with the directory or object
- See or modify the ACL of a dictionary or object
- Use the given name of a dictionary directory in the path name of another directory or an object

CDD privileges are governed by an inheritance principle. This means that each user with access to CDD\$TOP has access to every descendant of CDD\$TOP unless his or her access privileges are explicitly modified. ACLs at each dictionary directory below can modify inheritance by specifically granting or denying privileges to users or groups of users. These users inherit the modified privileges as they move down the dictionary path.

When the CDD is first installed, all users have all access privileges to CDD\$TOP. The person who manages the CDD on your system modifies these according to the needs of your installation so that users can get to the directories containing information they have a right to use, but cannot get to directories containing information they should not use.

Table 7-3 lists the access privileges users can have and describes what each allows you to do.

Table 7-3: Access Control Privileges

Privilege	Description
C (CONTROL)	Lets you read, create, modify, and delete access control list entries. You cannot deny yourself CONTROL privilege.
D (LOCAL_DELETE)	Lets you delete dictionary objects, as well as directories and subdictionaries with no children, and to edit, replace, or recompile definitions stored in the CDD.
E (DTR_EXTEND/EXECUTE)	Lets you ready a domain for any type of access, to access a table, or to invoke a procedure.
F (FORWARD)	Lets you create subdictionaries.
G (GLOBAL_DELETE)	Lets you delete dictionary directories and subdictionaries, including any children they may have, with a single command.
H (HISTORY)	Lets you add entries to CDD history lists with the Dictionary Management Utility (DMU).
M (DTR_MODIFY)	Lets you ready a domain for READ and MODIFY access.
P (PASS_THRU)	Lets you use a dictionary directory, subdictionary, or object in a path name. You cannot deny yourself PASS_THRU privilege.
R (DTR_READ)	Lets you ready a domain for READ access, display CDD definitions with a SHOW command and copy them into a command file with an EDIT or EXTRACT command.
S (SEE)	Lets you see the definition of a dictionary object. SEE access to a domain definition and its associated record definition is necessary to define a data file and then to ready the domain.
U (UPDATE)	Lets you update the definition of a dictionary object.
W (DTR_WRITE)	Lets you ready a domain for WRITE access.
X (EXTEND)	Lets you create children of dictionary directories and subdictionaries.

Refer to the CDD documentation for an explanation of history lists and subdictionaries.

7.9.1 Displaying Your Privileges

You can display the privileges you have for a directory or object by typing **SHOW PRIVILEGES FOR** followed by its path name and pressing **RETURN**. A simple **SHOW PRIVILEGES** entry displays the privileges you have for the directory at which you are currently located:

```
DTR) SHOW PRIVILEGES
```

```
Privileges for CDD$TOP.DTR$USERS.BELL
```

```
R (DTR_READ)           - may ready for READ, use SHOW and EXTRACT
W (DTR_WRITE)          - may ready for READ, WRITE, MODIFY, or EXTEND
M (DTR_MODIFY)         - may ready for READ, MODIFY
E (DTR_EXTEND/EXECUTE) - may ready to EXTEND, or access table or procedure
C (CONTROL)            - may issue DEFINEP, SHOWP, DELETEP commands
D (LOCAL_DELETE)      - may delete a dictionary object
F (FORWARD)            - may create a subdictionary
G (GLOBAL_DELETE)     - may delete a directory and its descendents
H (HISTORY)            - may add entries to object's history list
P (PASS_THRU)         - may use given name of directory or object in path name
S (SEE)                - may see (read) dictionary
U (UPDATE)             - may update dictionary object
X (EXTEND)             - may create directory or object within directory
```

```
DTR) SHOW PRIVILEGES FOR FAMILIES
```

```
Privileges for CDD$TOP.DTR$USERS.BELL.FAMILIES
```

```
R (DTR_READ)           - may ready for READ, use SHOW and EXTRACT
W (DTR_WRITE)          - may ready for READ, WRITE, MODIFY, or EXTEND
M (DTR_MODIFY)         - may ready for READ, MODIFY
E (DTR_EXTEND/EXECUTE) - may ready to EXTEND, or access table or procedure
C (CONTROL)            - may issue DEFINEP, SHOWP, DELETEP commands
D (LOCAL_DELETE)      - may delete a dictionary object
F (FORWARD)            - may create a subdictionary
G (GLOBAL_DELETE)     - may delete a directory and its descendents
H (HISTORY)            - may add entries to object's history list
P (PASS_THRU)         - may use given name of directory or object in path name
S (SEE)                - may see (read) dictionary
U (UPDATE)             - may update dictionary object
X (EXTEND)             - may create directory or object within directory
```

```
DTR)
```

The **SHOW PRIVILEGES FOR** command is useful when you need to use definitions and directories that are not your own. If you try an operation and receive an insufficient privilege message, you can check your privileges for the directories and definitions you need to use against the requirements in Table 7-4.

Table 7-4 tells you what privileges you must have to use the various **DATATRIEVE** commands and statements.

Table 7-4: Access Privilege Requirements for Commands and Statements

To Enter:	You Need in the ACL of the:	The Following Privileges:
DEFINE DICTIONARY DEFINE DOMAIN DEFINE PORT DEFINE PROCEDURE DEFINE RECORD DEFINE TABLE	parent directory	P (PASS_THRU) X (EXTEND)
DEFINE FILE	domain definition	P (PASS_THRU) S (SEE) W (DTR_WRITE)
	record definition	P (PASS_THRU) S (SEE) E (DTR_EXTEND/EXECUTE)
DEFINEP	definition or directory	P (PASS_THRU) C (CONTROL)
DELETE	parent directory	P (PASS_THRU) X (EXTEND)
	definition	P (PASS_THRU) and either D (LOCAL_DELETE) or G (GLOBAL_DELETE)
DELETEP	definition or directory	P (PASS_THRU) C (CONTROL)
EDIT path-name (You also need the privileges required to use the REDEFINE command and, if present, the DELETE command in order to exit the editor.)	parent directory definition	P (PASS_THRU) P (PASS_THRU) S (SEE) R (DTR_READ)

(continued on next page)

Table 7-4: Access Privilege Requirements for Commands and Statements (Cont.)

To Enter:	You Need in the ACL of the:	The Following Privileges:
EXECUTE (:) procedure-name	procedure definition	P (PASS_THRU) S (SEE) E (DTR_EXTEND/EXECUTE)
EXTRACT	definition	P (PASS_THRU) S (SEE) R (DTR_READ)
IN, NOT IN, or VIA dictionary-table-name	table definition	P (PASS_THRU) S (SEE) E (DTR_EXTEND/EXECUTE)
IN, NOT IN, or VIA domain-table-name	table definition	P (PASS_THRU) S (SEE) E (DTR_EXTEND/EXECUTE)
	domain definition	P (PASS_THRU) S (SEE) and either R (DTR_READ) or W (DTR_WRITE) or M (DTR_MODIFY)
	record definition	P (PASS_THRU) S (SEE) E (DTR_EXTEND/EXECUTE)
REDEFINE	parent directory	P (PASS_THRU) X (EXTEND)
	definition	P (PASS_THRU) S (SEE) R (DTR_READ) U (UPDATE)
READY (for all access modes)	record definition	P (PASS_THRU) S (SEE) E (DTR_EXTEND/EXECUTE)
READY... READ	domain definition	R (DTR_READ) or W (DTR_WRITE) or M (DTR_MODIFY) or S (SEE)

(continued on next page)

Table 7-4: Access Privilege Requirements for Commands and Statements (Cont.)

To Enter:	You Need in the ACL of the:	The Following Privileges:
READY... WRITE	domain definition	W (DTR_WRITE) S (SEE)
READY... MODIFY	domain definition	M (DTR_MODIFY) W (DTR_WRITE) S (SEE)
READY... EXTEND	domain definition	E (DTR_EXTEND/EXECUTE) or W (DTR_WRITE)
SET DICTIONARY	directory	P (PASS_THRU)
SHOW path-name	definition	P (PASS_THRU) S (SEE) R (DTR_READ)
SHOWP	definition or directory	P (PASS_THRU) C (CONTROL)

7.9.2 Displaying and Changing an ACL

You display an ACL with the SHOWP command. You can change an ACL with the DELETEP and DEFINEP commands. As Table 7-3 indicates, the ACL must assign you the C (CONTROL) and P (PASS_THRU) privileges for you to display or change the ACL. The CDD does not let you deny yourself these privileges when you already have them, but you cannot give these privileges to yourself if you do not have them.

The following example shows how you can modify the ACL for your top-level private directory to deny access to other users. If your top-level directory was created by the NEWUSER program, as was the BELL directory in the following example, it does not have an ACL. If this is the case, you must create one. (If you forget what your UIC is, you can exit DATATRIEVE and enter SHOW PROCESS at the dollar sign (\$) prompt. The display shows your UIC.)

```
DTR> SHOW DICTIONARY
The default directory is CDD$TOP.DTR$USERS.BELL

DTR> SET DICTIONARY CDD$TOP.DTR$USERS
DTR> SHOWP BELL
DTR> ! Return to the DTR> prompt means there is no ACL
DTR> DEFINEP FOR BELL 1 USER=BELL, UIC=[311,210], GRANT=ALL
DTR> DEFINEP FOR BELL 2 UIC=[*,*], DENY=ALL
DTR> SHOWP FOR BELL
  1:          [311,210], Username: "BELL"
      Grant - CDEFGHMPRSUX, Deny - none, Banish - none
  2:          [*,*]
      Grant - none, Deny - CDEFGHMPRSUX, Banish - none

DTR>
```

The important thing to remember when you create or modify ACLs is that each entry (identified as 1:, 2:, and so forth in the SHOWP display) is evaluated in the order that it appears. As soon as a match is found for any particular user, the CDD stops reading the list. It is very important, therefore, that the DEFINEP 1 command grants you your privileges and the DEFINEP 2 command denies privileges to other users. If you define a new entry 1, any existing entries are reassigned higher numbers so that they appear following 1.

After you get the top two entries correctly in place, you can use DELETEP to clean up any entries below 2 that you might have entered incorrectly or that the second entry supersedes. If the ACL for the BELL directory contained ACL entries numbered 3 and 4, for example, Bell could get rid of them with the commands DELETEP BELL 4 and DELETEP BELL 3.

Protecting the top-level directory of your private branch of the CDD also protects subordinate directories and objects. In addition, remember that you can protect data files in your VMS directories with the DCL SET PROTECTION command.

If you are interested in providing more extensive or specific protection, refer to the *VAX DATATRIEVE Reference Manual*. You should also read the next section if you plan to create an application that will be used by people in addition to yourself.

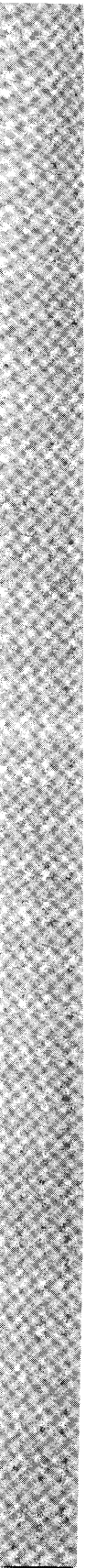
7.10 Using the CDD to Design Department-Wide or System-Wide Applications

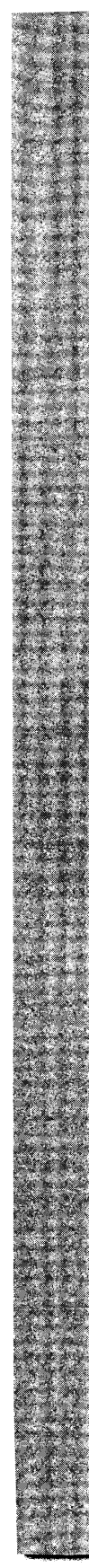
The CDD utilities provide more options for directory organization and maintenance and for access control than you can achieve using DATATRIEVE. Using CDD utilities, for example, you can organize a branch of the CDD as a *subdictionary* and assign it to a disk that you can remove from a disk drive for maximum security. You must also use CDD utilities to inspect and maintain auditing information (history lists). The CDD DMU utility also has an ACL editor that simplifies creation and maintenance of ACLs.

When you are designing applications that other people will use, you should refer to the CDD documentation for a complete explanation of CDD organization and use.



Part III
Setting Up an Application





Application Case Study: A Personnel System **8**

This chapter helps you analyze the requirements for a DATATRIEVE application so that you can translate those requirements into DATATRIEVE code. The sample application is a personnel system for an engineering firm. You will follow these steps:

1. Review the requirements
2. Analyze the data
3. Organize the data into domains and tables

8.1 Reviewing the Requirements

The following pages contain the Data Requirements Study for the sample Personnel System. When you design your own application, its requirements will likely be different from those for this application. However, you should follow the same procedure. Define your application's purpose and sketch out the data requirements it has. You can use the requirements categories in the sample study as a guide. Later, as you design your database and procedures, periodically review your requirements to make sure you do not forget to include any of them.

Data Requirements Study Personnel System

Purpose

All personnel systems must maintain employee data. Most systems must also answer online inquiries and create reports. The system for this personnel database must do all of these tasks.

System Requirements

System requirements relate to the devices that your application will be receiving data from and sending data to. System requirements also take into account whether or not your application will be receiving and sending information across a computer network:

- For data entry: All data will be entered at the terminal.
- For reports: Reports will be displayed at a video terminal or printed at a hardcopy terminal or printer.
- For distributed processing: This system will be autonomous. It will not share data with other computer systems.

Report Requirements

It is important to decide what kinds of reports your application must generate on a routine basis. What information your database contains depends to a large extent on the reports you want it to produce. This personnel system must generate the following reports:

- Individual employee report: Given an employee, list the detailed data pertaining to him or her. For example, provide all information about Nanette Greeb.
- Employee listings: Given a field or combination of fields in the employee record, list all the employees by that field. For example, list all employees by department, manager, or job title.
- Job category report: List all the job categories. Show the following information:
 - Job code and job title
 - Salary range

- Average actual salary for employees in the category
- Names of employees in the category
- Actual salary and wage class for each employee
- Department report: List employees, job titles, salaries, and dates of last performance reviews by department. This report is intended for department managers.
- Salary and job history: List employees, all the jobs they have held in the company, and the dates of their performance reviews.
- Educational background: List the college training completed by an employee, colleges attended, degrees and the dates they were received, and degree fields.
- Miscellaneous reports: Provide small, ad hoc reports generated from the personnel list format, such as address lists.

Online Inquiry Requirements

Online inquiry to a personnel database must be restricted to information that the person making the inquiry has a right to see. In this system, the following employees can access the information listed:

- Supervisors and department managers can access data that applies to their subordinates.
- Other employees can access only the names, job titles, and departments of company personnel.

Database Updating Requirements

Requirements for data update include how the data is maintained and how the system ensures the data is valid. This personnel system has the following updating requirements:

- Online maintenance: Personnel department employees will add, delete, or modify employee records on line. The system does not need to process transaction files to update the information stored in the domains.
- Automatic validation: The system must provide a way to make sure that department codes are valid and there are no duplicate employee identification numbers.

A real personnel system would include requirements relating to tax computation and benefits. These requirements are omitted from the sample system in this book so that you will find it easier to see the relationships among the domains in the system.

8.2 Analyzing the Data

At this stage of your application, you want to generate a list of the pieces of information your database should contain. There are a number of ways you can do this, but you might find it easiest to follow these steps:

1. Sketch out what you expect the reports to look like. The fields in the reports determine to a large extent the fields you will store in records.
2. Some fields depend on other fields. That is, there is a one-to-one correspondence between them. For example, every job code is associated with only one job title. Identify these fields. You might be better off storing these paired values in a DATATRIEVE table and putting only the smallest or key value in a record.
3. Some fields can be calculated from other fields. For example, age can be calculated from birth date. Fields like average salary and salary mid-points for a job category can be calculated from existing salaries and minimum and maximum salaries.

You can specify a field calculated from others in the same record as a **COMPUTED BY** field in the record definition. **COMPUTED BY** fields do not take up storage space because their values are calculated at the time you access a record. (**COMPUTED BY** fields are discussed in Chapter 9). If the calculated field appears in only one report, however, you might decide to create it as part of the procedure that produces the report rather than specifying it in a record definition.

Your goal at this point is to determine the minimum number of fields that you want to put in a record definition and which fields you want to take up space in storage.

4. Compile a list of data fields. Next to each field you might note the following information (if it applies to that field):
 - Any field with which it has a one-to-one correspondence
 - Any fields from which it can be calculated
 - Whether each value stored in the field must be unique
 - What makes values for the field valid ones

5. Determine the most efficient way to organize the fields into domains and tables.

Table 8-1 shows a list of fields you might start with when creating a personnel system. A number of fields would appear in more than one report. Some of them would probably never appear in the same report together. At this point, you want to know how many pieces of data you have to work with rather than how you are going to group them.

Table 8-1: Fields for Personnel System

Field	Unique?	Depends On	Valid If:	Calculated?
EMPLOYEE_ID	Yes	-	5 digits	-
LAST_NAME	-	-	-	-
FIRST_NAME	-	-	-	-
MIDDLE_INITIAL	-	-	-	-
ADDRESS_DATA	-	-	-	-
EMP_STREET	-	-	-	-
EMP_TOWN	-	-	-	-
EMP_STATE	-	-	-	-
EMP_ZIP	-	-	-	-
SEX	-	-	M or F	-
SOCIAL_SECURITY	Yes	-	-	-
BIRTHDAY	-	-	Valid date	-
JOB_CODE	Yes	-	-	-
JOB_TITLE	-	JOB_CODE	-	-
MINIMUM_SALARY	-	-	-	-
MAXIMUM_SALARY	-	-	-	-
SALARY_MIDPOINT	-	-	-	Min and Max Salary
WAGE_CLASS	-	-	-	-
DEPARTMENT_CODE	Yes	-	-	-
DEPARTMENT_NAME	-	DEPARTMENT_CODE	-	-

(continued on next page)

Table 8-1: Fields for Personnel System (Cont.)

Field	Unique?	Depends On	Valid If:	Calculated?
JOB_START	-	-	Valid date	-
JOB_END	-	-	Valid date	-
REVIEW_CODE	-	-	-	-
SALARY_AMOUNT	-	-	-	-
SALARY_START	-	-	Valid date	-
SALARY_END	-	-	Valid date	-
REVIEW_DATE	-	-	Valid date	-
SUPERVISOR_ID	-	-	-	-
DEGREE	-	-	-	-
DEGREE_FIELD	-	-	-	-
DATE_GIVEN	-	-	Valid date	-
COLLEGE_NAME	-	-	-	-

Expect that field requirements will change as you think about organizing them into domains or tables. This list of fields does not take into account, for example, special fields to indicate whether a record contains current or historical information.

8.3 Grouping Fields into Domains and Tables

After you know what fields you will need for your application, you have to decide how best to group them.

The simplest way to go about this is to define a record to match each report you want to produce and create a data file and domain to go with each record definition. This can work well for a small application whose requirements are not going to change and that will only be used by one or two people to generate formal reports.

If you are designing a database to support formal and informal reports and interactive queries, however, that is an inefficient solution. No number of data files is likely to meet the needs of all users. In addition, the organize-by-report method probably requires storing some fields in several data files. Each time an employee's name or address changes, for example, these field values have to be changed in every file that stores them.

Another method you might consider is to create one massive domain that contains every possible piece of information you would include in any report or query. This method, too, can create storage and maintenance problems. Besides, the results of statements like `PRINT domain-name` or `SHOW FIELDS FOR domain-name` would require a fast finger on the `NO SCROLL` key for people privileged to see all the information.

`DATATRIEVE` gives you a variety of methods to look at data stored in different locations. You should therefore pay special attention to ease of maintenance and logical grouping of fields when you put together a record.

Aim to put a field in only one place, unless you plan to use it as a link to related information stored somewhere else. Employee names, for example, are best stored in only one place. Employee ID numbers, however, probably need to be stored in several locations.

When you group fields together, consider grouping fields that contain generic data apart from fields that contain specific data. Job information, for example, can be generic (the same for each job code) or employee-specific. Generic information, such as wage class and minimum salary, can go in one domain. Employee-specific information, such as start date and review code, can go in another domain. If you keep generic data apart from specific data, you save storage space. If job entries for employees include wage class and minimum salary, values for these fields will be stored in many records when they need to be stored in only a few.

Figure 8-1 shows one way you could organize the fields in the sample personnel system. Above each grouping is the domain name that will eventually associate the record definition describing the fields with the data file that will store them.

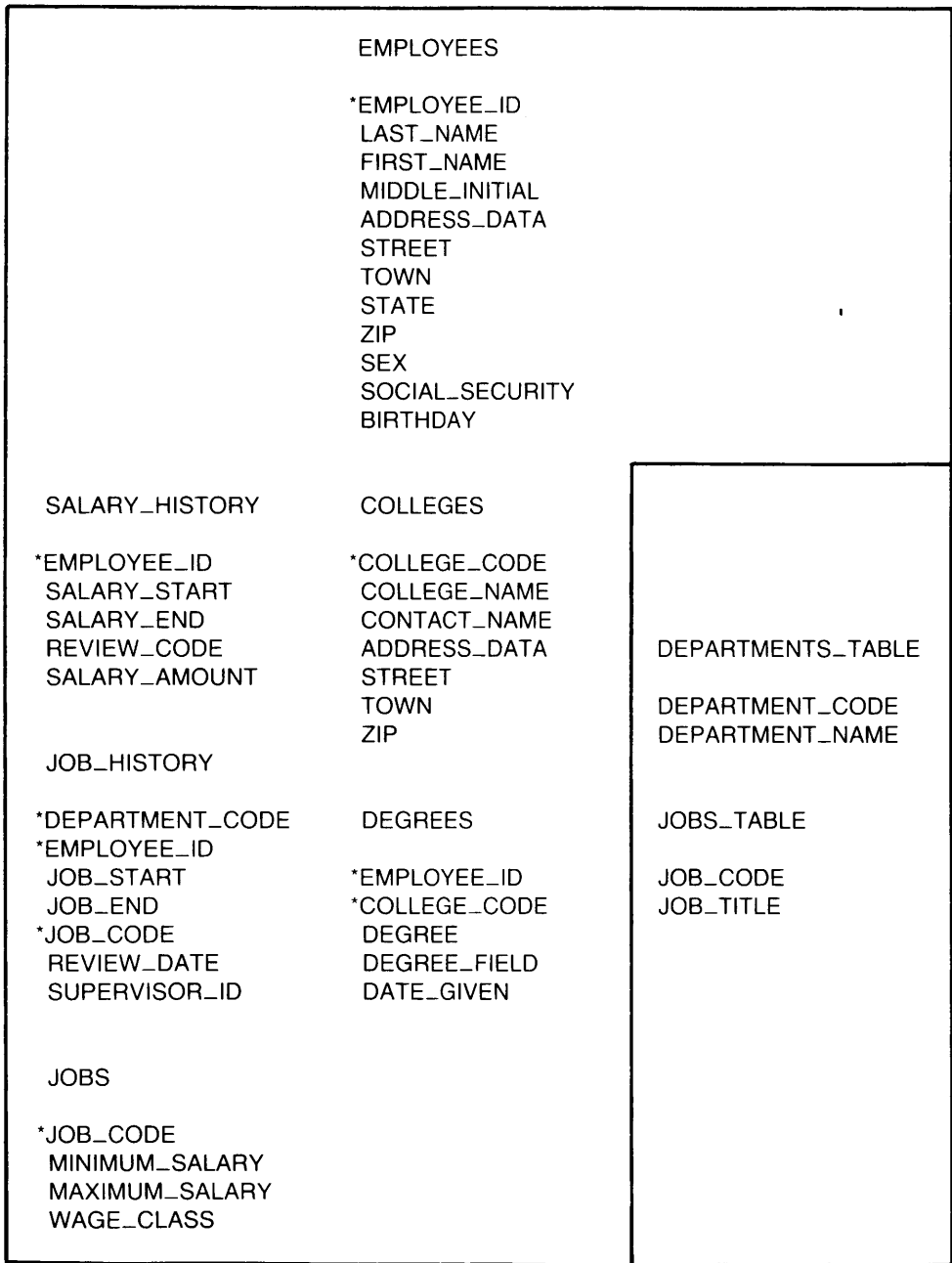
The fields preceded by an asterisk (*) indicate fields that you can use in `DATATRIEVE` statements to link data in one domain with data in the other domains. When grouping fields into domains for your own applications, you should note the following points about pivotal fields like these:

- They are the only fields that are stored in more than one place
- They are codes that can easily be made unique (and, unlike names, can stay that way). Many are likely to be primary keys for the data files to which they correspond. You cannot modify the value of primary key fields.

- Their values can be a set number of characters. It is easier to write statements that can check for valid values in fields that are always a set number of ordered characters.

Note

The sample personnel system outlined in this chapter is the basis for many examples in following chapters. Note, however, that the data definitions and data shown in examples are not included as part of DATATRIEVE and are *not* related to the sample PERSONNEL domain created during installation of DATATRIEVE. Although the structure of the sample database described here is similar to the sample Rdb/VMS database, PERSONNEL, created during the DATATRIEVE installation, you cannot reproduce the examples in this book by using that database.



*Denotes key fields

ZK-00002-00

Figure 8-1: Domains and Tables in Sample Personnel System



Defining Records 9

After you decide what fields you want to associate with a domain, you can create a record definition to describe them. If you worked through the examples in Chapter 1, ADT created a record definition for you. This chapter explains how you use the `DEFINE RECORD` command that hides behind ADT's menu of choices. There are several reasons why you might want to know the explicit way to define a record:

- If you want to change the record that ADT creates for you, you have to know what to edit and the consequences of changing anything.
- If you want to create a record definition to use with a data file that already exists, one created by a COBOL program perhaps, you have to match field definitions to the way they are stored.
- ADT does not give you all the `DATATRIEVE` options you can include in a record definition. The `VALID IF` and `COMPUTED BY` clauses are two examples of options you cannot include using ADT.
- If you are an experienced programmer and have put together many record definitions in the past, you want more specifics about what you are defining than the ADT menu provides. If you fall in this category, you probably can focus on the examples and tables in this chapter and simply skim through the explanatory text.

The best way to learn how to put a record definition together is to look at an existing one and pick it apart. Example 9-1 defines a record definition to go with the `EMPLOYEES` domain from the personnel system discussed in Chapter 8.

Comment lines starting with an exclamation point (!) provide labels and explanatory text that you would not enter if typing the DEFINE command. The sections following the example more fully explain the labels.

Example 9-1: Sample DATATRIEVE Record Definition

```

DTR> DEFINE RECORD EMPLOYEES_REC USING
DFN> !      ^           ^           ^
DFN> ! required   name of   optional
DFN> ! keywords  definition keyword
DFN> !
DFN> 01      EMPLOYEES_REC .
DFN> !      ^           ^           ^
DFN> ! level    name of   end of field
DFN> ! number   top-level field definition
DFN> !
DFN>          05      EMPLOYEE_ID PIC X(5)
DFN> !          ^           ^           ^
DFN> !          level  name of   field defini-
DFN> !          number field    tion clause
DFN> !
DFN>                                QUERY_NAME IS ID
DFN> !                                ^
DFN> !                                field defini-
DFN> !                                tion clause
DFN> !
DFN>                                QUERY_HEADER IS "EMP"/"ID" .
DFN> !                                ^           ^
DFN> !                                field defini-   end of field
DFN> !                                tion clause   definition
DFN> !
DFN>          05 EMPLOYEE_NAME          QUERY_NAME IS NAME.
DFN> !          ^
DFN> !          This group field contains the three elementary fields
DFN> !          that follow it.
DFN> !
DFN>          10 LAST_NAME          PIC X(14)
DFN> !                                QUERY_NAME IS L-NAME
DFN> !                                QUERY_HEADER IS "LAST NAME".
DFN>          10 FIRST_NAME         PIC X(10)
DFN> !                                QUERY_NAME IS F-NAME
DFN> !
DFN> !                                QUERY_HEADER IS "FIRST NAME".

```

```

DFN>          10 MIDDLE_INITIAL      PIC X
DFN>          QUERY_NAME IS INIT
DFN>          QUERY_HEADER IS "I".
DFN> !
DFN> !      ^
DFN> !      Note that all fields subordinate to EMPLOYEE_NAME
DFN> !      have level numbers with larger values.
DFN> !
DFN>          05 EMPLOYEE_ADDRESS.
DFN>          10 ADDRESS_DATA          PIC X(20).
DFN>          10 STREET                 PIC X(25).
DFN>          10 TOWN                   PIC X(20).
DFN>          10 STATE                  PIC X(2).
DFN>          10 ZIP                    PIC X(5).
DFN>          05 SEX                    PIC X
DFN>          VALID IF SEX = "M" OR
DFN>          SEX = "F".
DFN>          05 SOCIAL_SECURITY        PIC X(9)
DFN>          QUERY_NAME IS SS
DFN>
DFN>          EDIT_STRING XXXBXXBXXXX
DFN>
DFN>          VALID IF SS BETWEEN
DFN>          "1" AND "999999999".
DFN>          05 BIRTHDAY               USAGE DATE
DFN>          EDIT_STRING IS NN/DD/YY.
DFN> ;
DTR> ! ^
DTR> ! end of record definition
DTR> !
DTR>

```

9.1 Setting Up Relationships Among Fields (Level Numbers)

When you create a logical model for your record, you decide which fields will contain other fields and which fields will not. Figure 9-1 shows the logical model of fields in the sample employee record before they were defined:

```

EMPLOYEE_ID
EMPLOYEE_NAME
  LAST_NAME
  FIRST_NAME
  MIDDLE_INITIAL
EMPLOYEE_ADDRESS
  ADDRESS_DATA
  STREET
  TOWN
  STATE
  ZIP
SEX
SOCIAL_SECURITY
BIRTHDAY

```

Figure 9-1: Logical Model of EMPLOYEES__REC

The fields that contain other fields are **group fields**. In `EMPLOYEES_REC`, for example, `EMPLOYEE_NAME` and `EMPLOYEE_ADDRESS` are group fields. All the other fields in this logical model are **elementary fields**; they do not contain other fields. As you read down the list of fields, you can tell by the indentation in the figure which fields are included under a group heading and which are not. You can see, for example, that `SEX`, `SOCIAL_SECURITY`, and `BIRTHDAY` are not part of the `EMPLOYEE_ADDRESS` group.

You get `DATATRIEVE` to recognize the logical organization of your record by the level numbers you assign to the fields. When you assign level numbers, keep the following rules in mind:

- The first field in the record must be a top-level field that has a level number lower than that of any other field in the record. (Top-level fields are discussed further in a later section of this chapter.)
- You define subordinate fields by giving them level numbers that are higher than the field or fields that contain them.
- You specify the end of a logical set of fields by assigning to the next field in the record a level number equal to or lower than the lowest level number of the group field that begins that set.

Looking at the record definition in Example 9-1, you can see how level numbers implement the logical model for the sample employee record.

The highest logical level of the record is the 01 level field, `EMPLOYEES_REC`. Because 01 is a lower value than any other level number in the record, the field `EMPLOYEES_REC` contains all the other fields in the record.

All fields on the second logical level have the level number 05. Most of these fields are elementary fields (`EMPLOYEE_ID`, `SEX`, `SOCIAL_SECURITY`, and `BIRTHDAY`) and two are group fields (`EMPLOYEE_NAME` and `EMPLOYEE_ADDRESS`).

All fields on the third logical level have the level number 10 (`LAST_NAME`, `FIRST_NAME`, `MIDDLE_INITIAL`, `EMPLOYEE_ADDRESS`, `STREET`, `TOWN`, `STATE`, and `ZIP`). In this particular record, all the fields with level number 10 are elementary fields.

As `DATATRIEVE` reads down the record, it knows that the `EMPLOYEE_NAME` group ends as soon as it encounters the 05 level number of the `EMPLOYEE_ADDRESS` field. It knows that the `EMPLOYEE_ADDRESS` group ends as soon as it encounters the 05 level number of the field `SEX`. It knows that the `EMPLOYEES_REC` field ends because it encounters the semicolon (;).

You can choose any integer value from 1 to 65 for a level number. The actual values you select are not important; only the relative values matter.

Now you know what DATATRIEVE requires from you as far as level numbers are concerned. The following list of guidelines will make your record definitions easier to read:

- Use indentation to line up fields that are on the same logical level. This makes your record definition easier for you and others to read.
- Put a leading 0 in front of single digit values. Use 01, for example, rather than 1. When all your level numbers are two characters, it is easier to keep the numbers aligned.
- Increase level number values by more than 1 as you move from one logical level to another. Doing this gives you room to add subordinate group fields to the record in the future, without the problem of readjusting level number values for the entire record. In the sample definition, for example, the level numbers progress from 01 to 05 to 10.

9.2 Selecting Names

The names you select for the record definition and for all of the record fields must adhere to the following rules. Each name:

- Must begin with a letter
- Can consist only of letters, digits, hyphens, and underscores
- Must not duplicate a DATATRIEVE keyword
- Must not contain blanks
- Must be from 1 to 31 characters long
- Must end with a letter or digit

Some valid DATATRIEVE names are:

EMPLOYEE

SALARY_NET

CATEGORY1

YEAR-TO-DATE-SALES-FOR-PRODUCTS

Some invalid DATATRIEVE names are:

SIZE

(duplicates a keyword)

1984-EARNINGS

(does not start with a letter)

EMPLOYEE NAME

(contains a blank)

PRICE_(\$/LB)

(contains invalid characters)

THIS_NAME_HAS_TOO_MANY_CHARACTERS

(self explanatory)

The same field name can appear more than once in a record definition. Duplicate field names, however, must belong to different group fields.

Note

When you specify a duplicate field name in your DATATRIEVE statements, you have to qualify it so that DATATRIEVE knows which of the fields in the record you want. A record could contain two fields called NAME, for example. If one were in the group field DEPT and the other in the group field PROJECT, you would have to type PRINT DEPT.NAME or PRINT PROJECT.NAME.

Now that you know the DATATRIEVE requirements for names, you can read the next four sections for further guidelines.

9.2.1 Differences Between Record Name and Top-Level Field

The name you type following DEFINE RECORD specifies the name under which the definition is stored in a dictionary directory. The only time you use the record name is when you want to do something with the definition itself—look at it, edit it, delete it, and so forth. You never use a record name in the DATATRIEVE statements that handle data.

The first field name in a record definition is always the top-level field, a field that includes all the other fields in the record. In most statements that handle data, you rarely need to specify the top-level field; simply specifying the domain name usually gets you all the fields in the record. In some complex statements, however, you might want to specify all the fields in the record when the syntax requires a field name. Typing the name of the top-level field in this situation can save you many keystrokes.

What does all this explanation mean to you now when you are only at the stage of defining records? It means you should specify the same name for the top-level field as you want for the record definition. You do not have to do this but it makes it easy to avoid mistakes later on.

Are there any conventions to follow when choosing record names? Yes. Many people specify record names by taking the name they want for the domain and adding `_REC`. That is the convention used in Example 9-1 and by ADT. Other people always use plural nouns for domain names and singular nouns for records. Following this convention, if you want `EMPLOYEEES` to be the name of your domain, specify `EMPLOYEE` as the name of your record. Whatever convention you decide on, use it consistently. That way you do not have to check the domain definition to get the name of the record every time you want to edit the record definition.

9.2.2 Query Names

Field names should be descriptive of the data stored in the field rather than abbreviations that are easy to type. That makes the record definition easy to follow and maintain.

`DATATRIEVE` lets you both describe fields adequately and also abbreviate names for speed and ease of use. Add a `QUERY_NAME` clause to an elementary field definition to specify a shorter name you can use in place of the field name when typing `DATATRIEVE` statements. Example 9-1 has several examples of the `QUERY_NAME` clause. The keyword `IS` is optional when you type a `QUERY_NAME` clause.

When you define a query name for a field, you can use the query name as a replacement for the field name in any `DATATRIEVE` statements or clauses that refer to the field.

9.2.3 Column Headers

When you display data, `DATATRIEVE` uses the names you choose for the elementary fields in the record definition as default column headers for the stored values. If you segment the field name with underscores or hyphens, `DATATRIEVE` automatically uses multiple lines for the column header. This way, each segment in the name appears on a separate line in the display.

You can change the default column headers by adding a `QUERY_HEADER` clause to your elementary field definition. Example 9-1 contains several examples of a `QUERY_HEADER` clause. The keyword `IS` is optional. The header you select must be enclosed in quotation marks. Use a slash (/) to indicate that the following header segment should appear on the next line ("`EMP`"/"`ID`").

When you specify a column header, you and other users lose the advantage of knowing what the field names are simply by looking at data displays.

As long as your field names are descriptive of the data in the field, the main reason you want to add a `QUERY_HEADER` clause to the record definition is to optimize use of line space in your display. Some descriptive field names are longer than the values in the field. In Example 9-1, `MIDDLE_INITIAL` is an example of such a field. `DATATRIEVE` must use seven columns of display space for the column header when the values under the header only require one character.

The following example illustrates how several fields from `EMPLOYEES_REC` would display without the `QUERY_HEADER` clauses in Example 9-1:

```
DTR> READY EMPLOYEES
DTR> PRINT ID, NAME, TOWN, STATE OF FIRST 5 EMPLOYEES
```

EMPLOYEE ID	LAST NAME	FIRST NAME	MIDDLE INITIAL	TOWN	STATE
00164	Toliver	Alvin	A	Chocorua	NH
00165	Smith	Terry	D	Chocorua	NH
00166	Dietrich	Rick		Boscawen	NH
00167	Kilpatrick	Janet		Marlow	NH
00168	Nash	Norman		Meadows	NH

```
DTR>
```

Now the same display with the `QUERY_HEADER` clauses in Example 9-1:

```
DTR> PRINT ID, NAME, TOWN, STATE OF FIRST 5 EMPLOYEES
```

EMP ID	LAST NAME	FIRST NAME I	TOWN	STATE
00164	Toliver	Alvin	A Chocorua	NH
00165	Smith	Terry	D Chocorua	NH
00166	Dietrich	Rick	Boscawen	NH
00167	Kilpatrick	Janet	Marlow	NH
00168	Nash	Norman	Meadows	NH

```
DTR>
```

9.2.4 FILLER Field Name

You can specify the keyword `FILLER` as the name of an elementary or group field. You might want to specify `FILLER` if you:

- Do not need certain fields in a data file for a particular application

- Want to control record display to mask certain data (*not* for security reasons, just for display purposes)
- Want to reserve space in the physical record of the data file for future use

The rules for defining fields named **FILLER** are the same as those for other fields. Unlike other fields, however, you can use the name **FILLER** for more than one field in the same group field.

Values in **FILLER** fields cannot be accessed by **PRINT**, **LIST**, **MODIFY**, **STORE**, **REPORT**, and **SUM** statements. The contents of **FILLER** fields can always be viewed, however, by specifying in a **DISPLAY** statement the name of any group field containing them. Because even first level elementary **FILLER** fields belong to the top-level field in the record, you should not use the name **FILLER** to mask sensitive data from users who should not see it.

9.3 Specifying Type and Size of Data

This section explains how you tell **DATATRIEVE** about storage criteria; that is, what kind of characters are stored in a field and the maximum number of characters allowed in that field.

Every time you define an elementary field in your record definition, you must specify either a **PICTURE** (**PIC**, for short) or **USAGE** clause to tell **DATATRIEVE** what kind of characters are stored in the field and how many characters can fit.

9.3.1 Specifying a **PIC** Clause

A **PIC** clause starts with the keyword **PIC** and ends with a string of picture characters. Although you type a space after the word **PIC**, you cannot put a space anywhere in the string of picture characters that follows.

If you look at Example 9-1, you can see that all the **PIC** clauses contain the character **X**, sometimes followed by a number in parentheses. The **X** indicates that the field can contain any text character, roughly equivalent to any character you can type with a typewriter keyboard. (You can, however, press some keys on a terminal keyboard that produce nontext characters.) The number in parentheses is a **repeat count**. For example, **X(20)** means that a maximum of 20 text characters can be stored in the field. A repeat count is an option generally used when defining fields longer than three characters. When defining shorter fields, most people type a picture string character for each character in the field; for example, **PIC X**, **PIC XX**, or **PIC XXX**.

Table 9-1 lists and describes all the characters you can use in a PIC clause except the parentheses and number to designate repeat count. If you have limited programming experience, focus on the characters X, 9, V, and S. You can use the X character to define fields, such as names, that need to contain a wide range of characters. You use the characters 9, V, and maybe S to define fields, such as salary amount, on which you want to perform arithmetic operations. The characters A and P are listed in case you encounter them in a record definition created by someone else.

Table 9-1: Picture String Characters

Field Class	Picture Character	Meaning
Alphabetic	A	Represents one alphabetic character in the field.
Alphanumeric	X	Represents one character in the field.
Numeric	9	Represents one digit in the field. You can specify from 1 to 31 digits for a numeric field.
	S	Indicates that a sign (+ or -) is stored in the field. A picture string can have only one S and it must be the leftmost character.
	V	Indicates an implied decimal point. The decimal point does not occupy a character position in the field, but DATATRIEVE uses its location to align data in the field. A picture string can contain only one V.
	P	Specifies a decimal scaling position. Each P represents a "distance" in digits from an implied decimal point. A P can appear at the right or left of the picture string. A V is unnecessary for any picture string containing a P.

9.3.1.1 Defining Alphanumeric (X) and Alphabetic (A) Fields — Alphanumeric (X) fields are best for just about all fields unless you want to use the field values in arithmetic calculations.

Most people avoid defining alphabetic (A) fields. You cannot store hyphens, commas, periods, or numbers in alphabetic fields. Notice, however, that some names contain these characters:

SMITH-JONES

ARCO, INC.

TEA-FOR-2 CATERING

If you have some programming experience, you might be interested to know that DATATRIEVE has three relational operators especially designed for accessing text field values: CONTAINING, NOT CONTAINING, and STARTING WITH. You can also use the standard operators such as EQUALS, BETWEEN, GREATER_THAN, LESS_THAN, and so forth to access text field values in a range.

9.3.1.2 Defining Numeric Fields — As you can see by looking at Table 9-1, you can be more specific about the format of fields that contain only numbers. Depending on what characters you combine in the string, the field can contain only positive values or both positive and negative values. It can contain only integers or both integers and numbers with fractions.

Table 9-2 explains how to use numeric picture strings better than a wordy explanation can.

Table 9-2: TABLE Relating Numeric Picture Strings to Stored Values

Picture String	You Can Store:	You Cannot Store:
999	1, 10, 999	1000, -21, 2.5, "2"
S9(4)	1000, -1000, 21	10000, 2.5, -3.41, A_B
9(4)V99	1000, 3.5, 9999.99	-2, 1.314, 99999, 50%
V99	.15, .9, .80	1.5, -.45, 22, . 2
S9V9(4)	-1.5347, 2, .7	-78, .78902, \$2.45
9(5)PPP	12345000, 2112000	123450000, 21123.999
PPP9(5)	.00012345, .0003999	12345, 1.3

The picture character 9 represents places where significant digits can appear. The picture character P represents a digit you consider nonsignificant. Only zero can logically occupy a P position. If someone stores 12345 in a field defined as PIC 99PPP, the value is stored as 12000.

As you can see from Table 9-2, you use either V or P as a character to mark the position of the decimal point, but V is the only character you can use to insert a decimal point between 9s in the string.

Note

You can add a SCALE clause to substitute for Ps or a V in the picture string. For example, you could substitute PIC 9(5) SCALE 3 for PIC 9(5)PPP, PIC 9(5) SCALE -8 for PIC PPP9(5), and PIC 9(5) SCALE -2 for PIC 999V99. Do not use a SCALE clause, however, when the picture string contains a V or a P.

If you are a beginning programmer, you will find it easiest to limit yourself to numeric picture strings containing only 9s for digit positions and a V for decimal point position. You should also start the string with S if the field must store both positive and negative values. Numeric fields defined this way are iconic. You can visualize where the decimal point is by looking for a V and see how big the field is by adding up all the 9s. If there is no V, or a rightmost V, it means the field defines an integer.

You can define numeric picture strings from 1 to 31 digits long. Length in digits is the sum of all the 9s (and Ps, if any) in the picture string. You can specify one more character (S) to represent the sign.

9.3.2 Specifying a USAGE Clause

Every field definition has a USAGE clause, even if you do not explicitly specify one. USAGE DISPLAY is the default. It is the only usage that can apply to alphabetic and alphanumeric fields, and the one that applies to numeric fields unless you tell DATATRIEVE otherwise.

When the storage criteria for a numeric field are defined only with a PIC clause (PIC S99V99, for example), that field has display usage. You can do arithmetic calculations on a numeric field with display usage with no loss of precision, as long as any resulting values can be represented by 31 or fewer digits. It takes the computer a little longer to perform arithmetic calculations using fields that are display numeric, but the time loss is negligible for simple business applications.

All the other USAGE options exist to give programmers greater precision when defining and handling fields that contain numbers. Some of these options define fields that a variety of languages can process. This is important when you plan to create a data file that will be accessed not only by DATATRIEVE users, but also by programs written in languages such as COBOL, BASIC, and FORTRAN. You can ignore the rest of this section and resume reading about date fields in the next section if you have limited programming experience and the following statements are true in your case:

- Thirty-one or fewer 9s gives you the precision you need for results of arithmetic operations.
- You want to keep your definitions as simple to use as possible.
- You do not have to create a record definition to match an existing data file. (If the data file already exists, it might include numeric fields defined with USAGE options other than display.)

Table 9-3 lists and describes all the USAGE types supported by DATATRIEVE. When two keywords are grouped together in the table, it means they are synonymous.

Table 9-3: USAGE Clause Options

Type of Data	USAGE Option	Size in Bytes	Range (in Decimal Values)
ASCII string	DISPLAY (nonnumeric)	Depends on PIC clause	Depending on PIC clause, 1 to 65535 characters
	DISPLAY (signed numeric)	Depends on PIC clause	Total number of integer and decimal digits cannot exceed 31 (-999,...999 to 999,...999)
	DISPLAY (unsigned numeric)	Depends on PIC clause	Total number integer and decimal digits cannot exceed 31 (0 to 999,...999)
	COMP-3 PACKED	Depends on PIC clause	Same as DISPLAY (signed numeric)
	COMP-5 ZONED	Depends on PIC clause	Same as DISPLAY (signed numeric)
Fixed point binary	COMP INTEGER	Depends on PIC clause, if any. Default is 2.	Depending on size, -128 to $2^{*63} - 1$

(continued on next page)

Table 9-3: USAGE Clause Options (Cont.)

Type of Data	USAGE Option	Size in Bytes	Range (in Decimal Values)
Floating point binary	BYTE	1	-128 to 127
	WORD	2	-32768 to 32767
	LONG	4	-2^{31} to $2^{31} - 1$
	QUAD	8	-2^{63} to $2^{63} - 1$
	DATE	8	Although stored as an 8-byte binary number, values are a special case.
	COMP-1 REAL	4	$\pm(10^{-(38)} \text{ to } 10^{38})$, approximately 7 decimal digits precision
	G_FLOATING	8	$\pm(10^{-(308)} \text{ to } 10^{308})$, approximately 15 decimal digits precision
	COMP-2 DOUBLE	8	$\pm(10^{-(38)} \text{ to } 10^{38})$, approximately 16 decimal digits precision
H_FLOATING	16	$\pm(10^{-(4932)} \text{ to } 10^{4932})$, approximately 33 decimal digits precision	

You can indicate decimal point position with a **SCALE** clause. To indicate two decimal digits for a field that stores a fractional number, for example, you could define the field as **USAGE REAL SCALE IS -2**.

For more detailed information about data types, refer to the CDD documentation. Remember, however, that **DATATRIEVE** does not support **USAGE OCTAWORD** or unsigned value ranges for fixed point binary fields.

9.3.3 Date Fields

If you are defining a field to store dates, specify `USAGE DATE` for that field. `DATATRIEVE`, for example, correctly stores any of the following entries in a `USAGE DATE` field:

`28/MAR/1946`

`MAR 28 1946`

`March 28, 1946`

`3/28/46`

When you display a date value, `DATATRIEVE` formats it by default as `DD-Mmm-YYYY`. Any of the preceding entries would be displayed as `28-Mar-1946`, unless you specified an `EDIT_STRING` clause to change this default.

A section later in this chapter explains how to specify an edit string for a date field. Chapter 18 gives you more information about using date fields.

9.3.4 Virtual (Computed) Fields

When a field you are defining can be calculated by other field values or by values stored in a `DATATRIEVE` table, you can define it as a virtual field. A virtual field does not occupy any space in storage and so can reduce the size of your data files. The field value is calculated when you access it with a `DATATRIEVE` statement.

Define a virtual field with the `COMPUTED BY` clause. The value computation can include the name of one or more fields in the record definition or it might be accessed in a `DATATRIEVE` table.

In the following example, `GROSS_SALARY` and `DEDUCTIONS` are fields that appear in the same record definition as `NET_SALARY`:

```
05 NET_SALARY COMPUTED BY GROSS_SALARY - DEDUCTIONS.
```

The following example specifies a value using `STATES_TABLE`, a dictionary table that pairs the 2-character state code with the full name of the state (`MA` with Massachusetts, for example):

```
10 STATE_NAME COMPUTED BY STATE VIA STATES_TABLE.
```

You can use the `COMPUTED BY` clause only to describe elementary fields.

Many COMPUTED BY fields are better defined as variables that use the values in a record rather than as fields in the record definition. (A variable is a field you can define as necessary to get a particular value you need for a display or report.) This is especially true if the value you want to calculate uses a constant (such as tax rate) that is likely to change.

If you find that, just to satisfy the needs of a virtual field you want to compute, you are adding fields to your record that are likely to change, then consider doing the calculation outside the record definition. Chapter 18 explains how to define and use variables.

9.3.5 Defining One Record Area in Different Ways Using the REDEFINES Clause

The COMPUTED BY clause defines a field that occupies no space in a record. The REDEFINES clause takes another look at storage space that already exists. In the following example, CODE_FOR_SOMETHING is a value that is generally displayed and stored as a unit; however, users sometimes need to identify sections of the field:

```
03 CODE_FOR_SOMETHING PIC X(6).
03 SEGMENT_THE_CODE REDEFINES CODE_FOR_SOMETHING.
   06 FIELD1          PIC X(3).
   06 FIELD2          PIC X(3).
```

Note that a field redefining another must follow the field it is redefining. Both fields must have the same level number. In addition, the REDEFINES clause must immediately follow the field name in the field definition.

To understand REDEFINES on a higher level, you must first understand something about group fields—DATATRIEVE always considers them alphanumeric. This means when you use a group field name in a statement, DATATRIEVE looks at all the fields in the group as though you had defined them with Xs. This is true even if you used 9s or a numeric USAGE clause to define every subordinate item. If you want to define a field as numeric, but also want it to contain subordinates, you must redefine the field. In the following example, PART_NUMBER is a numeric field that has been redefined in two ways to identify subordinate fields:

```
05 PART_NUMBER          PIC 9(10).
05 PART_NUMBER_PARTS REDEFINES PART_NUMBER.
   10 PRODUCT_GROUP     PIC 99.
   10 PRODUCT_YEAR      PIC 99.
   10 ASSEMBLY_CODE     PIC 9.
   10 SUB_ASSEMBLY      PIC 99.
   10 PART_DETAIL       PIC 999.
```

```

05 PART_NUMBER_GROUPS REDEFINES PART_NUMBER.
   10 PRODUCT_GROUP_ID PIC 9(4).
   10 PART_DETAIL_ID PIC 9(6).

```

The REDEFINES clause adds complexity to a record definition that beginning programmers should probably avoid. Redefining a numeric field as an alphanumeric field and then using the two fields properly can be tricky. In addition, if you change the record definition later on to add new clauses or fields, you have to be very careful not to disrupt the relationship between a field and its redefinitions.

9.3.6 Specifying Repeating Fields Using the OCCURS Clause

You can define a list field to specify multiple occurrences of its subordinate field or fields. A record storing information about a family, for example, can define a list field to store information about children. In the following record definition, KIDS is a list field:

```

DTR> SHOW FAMILY_REC
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>

```

If you display records defined this way, you can see that records vary in the number of values stored in the fields KID_NAME and AGE:

```

DTR> READY FAMILIES
DTR> PRINT FIRST 3 FAMILIES

```

FATHER	MOTHER	NUMBER KIDS	KID 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

```
DTR>
```

The field KIDS is a **variable occurrence list** because the number of values in each record for fields subordinate to KIDS depends on a value stored in another field in the record (NUMBER_KIDS). Variable occurrence list fields must be the last set of fields in the record definition. Therefore, you can define only one variable occurrence list field in a record definition.

You can also define a **fixed occurrence list**. In this case, the number of values in each record for fields subordinate to the list field is specified explicitly in the OCCURS clause itself. If FAMILY_REC were altered to define a fixed occurrence list, the definition for KIDS would be:

```
03 KIDS OCCURS 10 TIMES.
```

If you display records containing a fixed length list, "empty" occurrences occupy space in the display. This can take the form of blank lines between records (if all the list subordinates are text fields) or columns of zeros (under fields defined as numeric). The advantage of defining a list that is fixed length rather than variable length is that it does not have to be the last set of fields in the record definition. While it is not recommended, you can also define any number of fixed length lists within a variable length list.

Accessing fields subordinate to an OCCURS field takes time to master. It is also difficult to restructure a domain when you want to add to or reorganize the fields subordinate to a list field.

DATATRIEVE sees each set of list values as an inner record within the record. You must treat the field defined with the OCCURS clause as you would a domain name. If you tried to specify KIDS, for example, in a DATATRIEVE statement where you normally specify a field name, DATATRIEVE might tell you that KIDS is undefined or used out of context. The following example illustrates this problem and one way to get around it:

```
DTR> READY FAMILIES
DTR> PRINT FATHER, KID_NAME OF FAMILIES
"KID_NAME" is undefined or used out of context.
DTR> PRINT ALL FATHER, ALL KID_NAME OF KIDS OF FAMILIES
```

FATHER	KID NAME
JIM	URSULA
JIM	RALPH
	ANNE
	JIM
	ELLEN
	DAVID
	ROBERT
JOHN	ANN
	JEAN
JOHN	CTRL/C

^C

Execution terminated by operator.

DTR>

You may have to specify list fields in a record definition if you are trying to create a DATATRIEVE record definition for a data file that already exists. If you do specify an OCCURS field and it contains more than one subordinate item, be sure you specify a top group subordinate to the OCCURS item itself. EACH_KID is an example of such a field in FAMILY_REC. This gives you a group field name that you can use like a field name. Chapter 15 provides more information on accessing fields subordinate to list fields.

When setting up your own database, however, you should avoid list fields. The set of domains for the personnel system in Chapter 8 provides an example of how to do this. Each salary history and job history entry for an employee is stored as a separate record. These entries are kept out of the record in the central employee domain by putting them in separate domains. If you are wondering how you can uniquely identify each record from the SALARY_HISTORY and JOB_HISTORY domains, the answer is to define a group field key for the data file. Chapter 11 gives you more information on this topic.

9.4 Formatting the Display of Field Values

You can always specify the output format of an elementary field value by including an edit string in a PRINT statement. If you want this information in a record definition, you use the EDIT_STRING clause. Example 9-1 includes an edit string for the field SOCIAL_SECURITY (EDIT_STRING IS XXXBXXBXXXX) to display the value with a space between segments of the field. It also includes an edit string for the field BIRTHDAY to replace the DATATRIEVE default of DD-Mmm-YYYY (EDIT_STRING IS NN/DD/YY). This is how those fields would display without the edit string:

```
DTR> READY EMPLOYEES
DTR> PRINT NAME, SOCIAL_SECURITY, BIRTHDAY OF FIRST 1 EMPLOYEES
```

LAST NAME	FIRST NAME	SOCIAL SECURITY	BIRTHDAY
Toliver	Alvin	A 763080064	28-Mar-1947

```
DTR>
```

This is how those same fields display with the edit strings in Example 9-1:

```
DTR> READY EMPLOYEES
DTR> PRINT NAME, SOCIAL_SECURITY, BIRTHDAY OF FIRST 1 EMPLOYEES
```

LAST NAME	FIRST NAME	SOCIAL SECURITY	BIRTHDAY
Toliver	Alvin	A 763 08 0064	3/28/47

```
DTR>
```

If you do not supply an EDIT_STRING clause for a numeric field, DATATRIEVE uses the PIC clause to format the field value. If the PIC clause contains a V or P, DATATRIEVE displays the value with a decimal point in the correct position. You usually want to include an EDIT_STRING clause for numeric fields that:

- Include a fractional component and that do not indicate decimal point position in the PIC clause
- Include a sign that you want to display (even if there is one in the PIC clause)
- Store money values

The appendix at the end of this book lists and describes all the edit string characters you can use. Tables 9-4, 9-5, and 9-6 illustrate examples of using editing characters for text, numeric, and date fields. The symbol # in the output values in each of these tables represents a space.

Table 9-4: Editing Text Fields

Picture String	Edit String	Output If Field Value Is CHALLENGER	Output If Field Value Is 123
X(10)	X(10)	CHALLENGER	123#####
X(10)	X(3)	CHA	123
X(10)	XX/X(8)	CH/ALLENGER	12/3#####
X(10)	X(5)/X(5)	CHALL/ENGER	123##/#####
X(10)	X(5)-XX	CHALL-EN	123##-##

Table 9-5: Editing Numeric Fields

Picture String	Edit String	Field Content	Output
9(5)	9(5)	04092	04092
9(5)	Z(5)	04092	4092
9(5)	*(5)	04092	*4092
99V99	99.99	0001	00.01
99V99	ZZ.99	0001	##.01
99V99	(None)	1234	12.34
99V99	Z9.99	1234	12.34
99V99	999.9	1234	012.3
99V99	9.999	1234	*****
99V99	9.999	0123	1.230
99V99	Z(4)	1234	##12
S9999	None	-1234	1234
S9999	-9999	-1234	-1234
S9999	-9999	+1234	#1234
S9999	9999+	-1234	1234-

(continued on next page)

Table 9-5: Editing Numeric Fields (Cont.)

Picture String	Edit String	Field Content	Output
S9999	+9999	+1234	+1234
S9999	9999DB	-1234	1234DB
S9999	9999CR	-1234	1234CR
S9999	CR9999	+1234	##1234
S9999	((9999))	-1234	(1234)
99	99%	45	45%
9(6)	\$999,999	100000	\$100,000
9(6)	\$\$\$\$,\$\$\$	100000	\$100,000
9(6)	ZZZ,ZZZ	000040	#####40
9(6)	999/999	123456	123/456

Table 9-6: Editing Date Fields

Edit String	Output if Field Value is June 4, 1980	Output if Field Value is November 27, 1978
DD- <i>MMM</i> -YY	#4-Jun-80	27-Nov-78
<i>MMM</i> B <i>DD</i> BY(4)	Jun##4#1980	Nov#27#1978
M(9)B <i>DD</i> BY(4)	June##4#1980	November#27#1978
NN/DD/YY	#6/04/80	11/27/78
W(9)	Wednesday	Monday
YYYY/JJJ	1980/156	1978/331
DDB <i>MMM</i> BY/WWW	#4#Jun#80/Wed	27#Nov#78/Mon
DD.NN.YY	#4.06.80	27.11.78

9.5 Including Validation Requirements

You can specify a `VALID IF` clause to make sure a value is correct before it is stored in a record field. Example 9-1 includes `VALID IF` clauses for the fields `SEX` and `SOCIAL_SECURITY` to limit the values these fields can contain. Because these fields are text fields, the acceptable values are enclosed in quotation marks. Values for numeric fields would not be enclosed in quotation marks.

You can specify a `VALID IF` clause only for an elementary field.

9.6 Initializing Field Values

`DATATRIEVE` automatically initializes a text field to spaces and a numeric field to zero if you do not assign it a value when storing a record. If you want a field initialized to any other value, use the `DEFAULT VALUE` clause to specify your preference. Example 9-1 does not contain this clause.

One way to use `DEFAULT VALUE` is with date fields. If the field should reflect the date a record is stored, you can specify the value expression `"TODAY"` as its default value:

```
03 DATE_IN  USAGE DATE DEFAULT "TODAY".
```

As you can see from the example, the word `VALUE` is an optional keyword.

9.7 Specifying Values to Be Ignored in Statistical Computations

You can define a `MISSING VALUE` clause to mark that no value is stored in a field. `DATATRIEVE` ignores fields containing the missing value marker when computing averages, standard deviations, and minimum and maximum values.

Numeric fields are automatically initialized to zero if no value is stored in them. It is fairly common for records to be stored in incomplete form. If a field storing a salary contains zero, for example, it usually means that the salary data has not been stored, not that the employee is working solely for fun. If you are averaging the salaries of all current employees in a given job category, you do not want records with these "empty" salary fields to affect your results. You can include the `MISSING VALUE` clause in the field definition to make sure that salaries equal to zero are ignored:

```
05 SALARY_AMOUNT          PIC 9(6)V99
                           EDIT_STRING IS $$$$,$$$.$$
                           MISSING VALUE IS 0.
```

Do not use the `DEFAULT VALUE` clause along with the `MISSING VALUE` clause unless they specify the same value. If they specify different values, `DATATRIEVE` initializes an empty field to the default value and includes that value in statistical computations.

9.8 Ending Field and Record Definitions

You must end each field definition with a period. Be careful, however, not to put a period after a clause that is not the last clause in the field definition:

```
DTR> DEFINE RECORD TEST USING
DFN> 01 TOP_GROUP .
DFN> 05 FIRST_FIELD PIC 9
DFN> 05 SECOND_FIELD PIC X.
DEFINE RECORD TEST USING
01 TOP_GROUP .
05 FIRST_FIELD PIC 9
05 SECOND_FIELD PIC X.
^
```

Expected field option or period, encountered "05".
DTR> ! A period did not terminate the definition of FIRST_FIELD.
DTR> !

```
DTR> DEFINE RECORD TEST USING
DFN> 01 TOP_GROUP .
DFN> 05 FIRST_FIELD PIC 9 .
DFN> QUERY_NAME IS FF.
DEFINE RECORD TEST USING
01 TOP_GROUP .
05 FIRST_FIELD PIC 9 .
QUERY_NAME IS FF.
^
```

Expected number, encountered "QUERY_NAME".
DTR> ! Period should not follow PIC clause of FIRST_FIELD.
DTR>

DATATRIEVE also insists that you end a record definition with a semicolon (;). It will keep displaying the DFN> prompt until you enter one.

9.9 Editing Record Definitions

When you edit a record definition, you see the keyword REDEFINE where DEFINE used to be. The REDEFINE RECORD command follows the same rules as DEFINE RECORD. REDEFINE RECORD, however, automatically creates a new version of an existing record definition.

The DEFINE RECORD command works only when there is no record definition with the specified name in the dictionary directory.

If you want to modify a record definition that is being used with an existing data file, read the section on restructuring domains in Chapter 11.

Defining Domains 10

A domain is the essential part of DATATRIEVE. It:

- Relates a record definition to a data file
- Gives a name to that relationship (domain name)

After you create the record and domain definitions and define the data file, you use the domain name to access the file.

Note

You can also create domain definitions that point to data stored on other computer systems and in VAX DBMS and relational databases. Refer to the *VAX DATATRIEVE User's Guide* for information on using DATATRIEVE with distributed data and with VAX DBMS and relational databases.

Example 10-1 illustrates a sample session in which user Bell creates a domain definition. The definition relates the sample record definition from Chapter 8 (EMPLOYEES_REC) with the file EMPLOYEES.DAT. The domain name EMPLOYEES identifies this relationship. Comment lines starting with an exclamation point (!) tell you what user Bell is doing.

Example 10-1: Defining a Sample Domain

```
DTR> ! Set dictionary location to the directory that will store
DTR> ! the domain definition.
DTR> !
DTR> SET DICTIONARY CDD$TOP.PERSONNEL
DTR> !
DTR> ! Define the domain.
DTR> !
DTR> DEFINE DOMAIN EMPLOYEES USING
DFN> EMPLOYEES_REC ON EMPLOYEES.DAT;
DTR> !
DTR> ! Display the listing of domains in the directory PERSONNEL.
DTR> !
DTR> SHOW DOMAINS
Domains:
      EMPLOYEES;1

DTR> ! Display the domain definition.
DTR> !
DTR> SHOW EMPLOYEES
DOMAIN EMPLOYEES USING EMPLOYEES_REC ON EMPLOYEES.DAT;

DTR> ! Decide to make the file specification more complete.
DTR> !
DTR> EDIT EMPLOYEES
      :
      :
      :

DTR> SHOW EMPLOYEES
DOMAIN EMPLOYEES USING EMPLOYEES_REC ON DBA2:[BELL]EMPLOYEES.DAT;

DTR>
```

As you can see from the example, you use the `DEFINE DOMAIN` command to begin a domain definition. The keyword `USING` is optional. You must end a domain definition with a semicolon (;). The next three sections provide rules and suggestions for naming the domain and specifying the record and file.

10.1 Naming the Domain

The name you choose for a domain must follow the rules that apply to any given name in the CDD. The domain name:

- Must begin with a letter
- Must end with a letter or digit
- Can contain 1 to 31 characters
- Can contain only letters, digits, dollar signs (\$), underscores (_), and hyphens (-)

When you enter a name, DATATRIEVE interprets lowercase letters as uppercase and a hyphen as an underscore. You see names displayed using this format.

If you prefer, you can specify a full dictionary path name for a domain name. This lets you store a domain definition in a directory other than the one at which you are currently located. In Example 10-1, if user Bell had not first set his dictionary location to the PERSONNEL directory, he could have entered CDD\$TOP.PERSONNEL.EMPLOYEES in place of EMPLOYEES to place the domain in that dictionary directory. Because DATATRIEVE stored the EMPLOYEES definition in PERSONNEL, you know Bell must have at least P (PASS_THRU) and E (EXTEND) privileges in the ACL associated with that directory.

A full path name is part of the definition, however, and has to be edited if you or someone else moves the definition later on. Most people define a domain using only the given name.

10.2 Specifying the Record Name

The rules that apply to the record name are the same as those for the domain name. If the record definition is (or will be) in a dictionary directory other than the one where you are storing the domain definition, you must specify a full path name for the record definition. Otherwise, you can specify the given name.

If you are specifying a full path name for a record definition in a directory that is not in your private branch of the CDD, make sure you have P (PASS_THRU) and S (SEE) access privileges to that record definition. You do not need these privileges to put the path name in your definition, but you need them in order to ready the domain.

10.3 Specifying the Data File

The name of the data file is a VMS file specification. File names are not governed by the CDD rules for naming things; they follow VMS rules. Chapter 2 contains the rules that apply to file specifications. You should review this information if you have limited experience using the VMS operating system.

10.3.1 How Much of the File Specification to Include

The shortest form you can use for a file specification in a domain definition is a file name (EMPLOYEES, for example). When you use this short form, DATATRIEVE appends the file type .DAT to the name. When you ready a domain whose definition includes only a file name (or only a file name and type), DATATRIEVE expects to find the data file in your default VMS directory.

You can include a full file specification in place of the short form. If you do this, you can ready the domain without first setting your VMS directory default to the directory containing the file. In Example 10-1, user Bell changed the file specification to DBA2:[BELL]EMPLOYEES.DAT when he edited the domain definition.

Note

If your installation uses more than one computer system, a file specification can start with a node name. If you want to use a data file on another system, you might try to append a node name along with user name and password criteria to your file specification. Do not do this. DATATRIEVE works *very slowly* when you access distributed data this way. The chapter on accessing remote data in the *VAX DATATRIEVE User's Guide* explains better ways to access data on other computer systems.

10.3.2 Avoiding Problems When Naming Files

If you break one of the VMS rules governing file specifications, you can still store the domain definition. When you try to create the file with the DEFINE FILE command, however, you will get an error message from RMS telling you about the problem.

If you are creating the domain in order to use a data file that already exists and the file specification is incorrect, you will get an error message when you try to ready the domain. If the file is in a directory other than your own, you will need the appropriate VMS access privileges to both the directory where the file is located and to the file itself before you can ready the domain. Chapter 2 provides more information on directory and file protection.

Defining Data Files 11

Once you store domain and record definitions in a dictionary directory, you can create a file in a VMS directory to contain your data. You omit this step, of course, if you created domain and record definitions to access a data file that already exists.

You use the DEFINE FILE command to create a data file. Example 11-1 first displays domain and record definitions (EMPLOYEES and EMPLOYEES_REC) and then creates the data file (EMPLOYEES.DAT).

Example 11-1: Defining a Data File

```
DTR> SHOW EMPLOYEES
DOMAIN EMPLOYEES USING EMPLOYEES_REC ON DBA2:[BELL]EMPLOYEES.DAT;
```

```
DTR> SHOW EMPLOYEES_REC
RECORD EMPLOYEES_REC USING
01 EMPLOYEES_REC.
   05 EMPLOYEE_ID          PIC X(5)
                           QUERY_NAME IS ID
                           QUERY_HEADER IS "ID".
   05 EMPLOYEE_NAME       PIC X(14)
   10 LAST_NAME           QUERY_NAME IS NAME.
                           QUERY_HEADER IS "LAST NAME".
   10 FIRST_NAME          PIC X(10)
                           QUERY_NAME IS F_NAME
                           QUERY_HEADER IS "FIRST NAME".
   10 MIDDLE_INITIAL      PIC X
                           QUERY_NAME IS INIT
                           QUERY_HEADER IS "I".
   05 EMPLOYEE_ADDRESS    PIC X(20)
   10 ADDRESS_DATA        QUERY_NAME IS ADDRESS.
                           QUERY_HEADER IS "ADDRESS".
                           .
                           .
                           .
```

```
DTR> DEFINE FILE FOR EMPLOYEES KEY = ID (NO DUP)
DTR>
```


Unlike the other DEFINE commands, DEFINE FILE is not creating a definition. It does not, therefore, specify the name of the file, but points to the domain definition that does (EMPLOYEES). It also does not require the semicolon or END_ clause that must terminate other DEFINE commands. The keyword FOR, by the way, is optional.

Example 11-1 creates an indexed file because it specifies a field in the record (ID) as a key. If the command were simply DEFINE FILE FOR EMPLOYEES, it would have created a sequential file. The differences between indexed and sequential files are discussed in the next two sections.

11.1 Defining Indexed Files

In almost all cases, it is better to define an indexed file because:

- You can delete records only from an indexed file.
- Only indexed files have keys.

Record access is faster when you can specify a key field to help DATATRIEVE find a record. When DATATRIEVE cannot use a key field, it has to perform an exhaustive search through the file for the record or records you want.

You can define more than one key for an indexed file. If you do, the first key you specify is the primary key and the others are alternate keys.

You cannot change the value of a primary key field. For each alternate key, however, you can choose whether or not users can modify the value in the key field after a record is stored (CHANGE or NO CHANGE). CHANGE is a default key characteristic for alternate keys.

For both primary and alternate keys, you can choose whether or not users can store more than one record with the same value in the key field (DUP or NO DUP). NO DUP is a default key characteristic for primary keys and DUP is the default for alternate keys.

The following command explicitly specifies all key characteristics so you can see command format and punctuation. The characteristics specified are the DATATRIEVE defaults for primary and alternate keys:

```
DEFINE FILE FOR EMPLOYEES KEY = ID (NO CHANGE, NO DUP),  
      KEY = L_NAME (CHANGE, DUP)
```

The following command specifies only the key characteristic needed to change a default:

```
DEFINE FILE FOR EMPLOYEES KEY = L_NAME (DUP),  
KEY = STATE
```

L_NAME, a primary key, cannot be changed. State, by default, can be both changed and duplicated.

11.1.1 Selecting the Primary Key

The field you select for a primary key should be one whose values do not change. DATATRIEVE does not let users modify values in primary key fields. That is why primary key fields are so often a code of some kind: ID number, invoice number, customer code, product code, and so forth. The codes can remain constant even if someone decides to change the name or other characteristics associated with the record. If you *must* change a primary key value, you have to erase the whole record and store it again with the changed key value.

The primary key for a file should be able to uniquely identify each record. This means you should avoid the DUP characteristic for the primary key field even though DATATRIEVE lets you use it. There are two reasons for this guideline:

- If at any time in the future you want to modify the records in the file based on information contained in another data file (transaction file processing), you will probably need a record-to-record match. This is impossible to get if you cannot specify a field or group of fields that is common to both files and that identifies only one record that is in the file you are changing.
- Retrieving data using a key field that contains many duplicate values can slow DATATRIEVE response time. The primary key is the one you will be using most often to associate records stored in more than one file. You will want this operation to proceed as quickly as possible.

If your application is limited to one small data file, go ahead and choose any field you want for a primary key and allow duplicate values if that is necessary.

Records are stored in ascending order according to the value of the primary key. The DEFINE FILE command in Example 11-1 specifies that records are stored so that the record with the lowest value for EMPLOYEE_ID is positioned first in the file and the record with the highest value for EMPLOYEE_ID is last in the file.

The order of the values in the primary key field is the default **sort order** for the data file. This is the order in which records are displayed when you simply type PRINT followed by the domain name. Chapters 14 and 15 discuss how you can change the sort order of records for a particular operation.

11.1.2 Selecting Alternate Keys

If you expect to frequently ask DATATRIEVE to search for records based on values in fields other than the primary key field, you can define those fields as alternate keys. A name associated with a record (LAST_NAME from the EMPLOYEES domain, for example) is one field often selected as an alternate key.

Do not specify as a key any field that may contain many duplicate values. A field such as SEX is an example of a poor key choice. When DATATRIEVE has to process many duplicate values, a key-based search can sometimes take longer than a sequential one.

Confine your alternate key choices to fields you expect to use frequently when retrieving records. From the file maintenance point of view, the fewer keys you define, the better.

11.1.3 Selecting Group Field Keys

You might want to select a group field key when no single elementary field can uniquely identify each record in the file.

Suppose that your file stores information about products manufactured by a number of vendors. For each vendor, there is more than one product and you cannot be sure that different vendors select differing product codes. If you need to ensure that one field identifies only one product, you can specify as a primary key a group field (PRODUCT_ID) that contains both vendor and product codes:

```
05 PRODUCT_ID .
  10 VENDOR_CODE      PIC X(5) .
  10 PRODUCT_CODE     PIC X(20) .
```

There are some restrictions when you specify group field keys:

- When you access the file using the key name, DATATRIEVE uses only the first elementary field in the key for indexed access.

Using the `PRODUCT_ID` example, DATATRIEVE directly finds the records with matching `VENDOR_CODE`, and then sequentially searches those records to find the matching `PRODUCT_CODE`. The partial sequential search through records means that access by group field key proceeds more slowly than access by elementary field key. The performance difference would probably bother only users who are trying to pull together large numbers of records.

- The first elementary field in the group must be alphanumeric, or DATATRIEVE does not do key-based access at all.

If `VENDOR_CODE` were defined as PIC 9(5), for example, DATATRIEVE would sort through records one by one to find the records you ask for. This performance lag could become very serious.

- You cannot specify a list field as a key.

There are three domains in the sample personnel system used in this book (`SALARY_HISTORY`, `JOB_HISTORY`, and `DEGREES`) that have no key that uniquely identifies each record. For any value of `EMPLOYEE_ID`, there can be more than one record in the file. The requirements for the personnel system specify that if records in these domains need to be updated, the updating operations should be done interactively. In the vast majority of cases, the maintenance operations involve adding new records to the domain rather than changing or deleting existing ones. In addition, these domains are generally accessed to pull together all the information for an employee or look at the current job and salary for one employee. Given these requirements, the keys perform a relational function and duplicate primary keys are not likely to cause problems.

11.2 Defining Sequential Files

Records in a sequential file are positioned in the order they are written to the file. If you type `PRINT` followed by the domain name, the first record displayed is the first one stored in the file. The last record displayed is the last one stored.

Sequential files have the following advantages:

- They are the only files you can store on magnetic tape.
- Report-writing procedures sometimes work more quickly when DATATRIEVE is processing records stored in a sequential file.

The disadvantages of creating sequential files outweigh the advantages:

- You cannot delete records from sequential files. You can approximate a delete operation for a sequential file by modifying all elementary field values to contain nothing but spaces or zeros. The “pseudo-erase” you must use with a sequential file, however, is time-consuming, wastes storage space and can show up in displays and reports as a blank line.
- Record update and data retrieval operations proceed more slowly because DATATRIEVE must always exhaustively search the file for each record you need.

11.3 Planning for File Maintenance

The term **file maintenance** in this section refers to the methods you employ to optimize file storage requirements and to improve DATATRIEVE response time. The discussion provides an overview of the topic and is not a detailed explanation. For more information, refer to the chapter on improving DATATRIEVE performance in the *VAX DATATRIEVE User's Guide* and to the documentation on record management services.

11.3.1 Using Other Options in the DEFINE FILE Command

You can determine the storage space assigned to your file by specifying:

- **ALLOCATION = n** to specify the number of disk blocks initially allocated for the file. If you understand how the size of your file relates to disk block requirements, you can use it to ensure that enough space is reserved on the disk to fit in all the records you plan to store. If you omit this clause, DATATRIEVE initially allocates 0 blocks for the file and adds blocks as records are stored.
- **SUPERSEDE** to eliminate a previous version of a file when you create a new one. For this option to take effect, you must include the number of the version you are superseding in the file specification contained in the domain definition. If you do not include the keyword **SUPERSEDE** or do not specify a version number, DATATRIEVE assigns the new file the next higher version number.
- **MAX** to specify that records containing variable length list fields occupy enough space in a sequential file to accommodate the maximum number of occurrences defined for the list field. This wastes storage space, but lets you add more list items to a record after storing it in a sequential file.

Here is an example of a DEFINE FILE command that contains all but the MAX option from this section. (The domain definition specifies the file DBA2:[BELL]FAMILY.DAT;1.) Note that you specify a KEY clause last:

```
DTR> DEFINE FILE FOR FAMILIES ALLOCATION = 30,  
SUPERSEDE,  
KEY = FATHER (DUP)
```

11.3.2 Using RMS Utilities to Load and Maintain Files

If you are creating large data files or indexed files that contain many keys, you should use Record Management Services (RMS) utilities to create and load your files and to periodically maintain them. Doing this can improve DATATRIEVE response time. Using RMS utilities, you can:

- Consolidate disk storage of data and indexes

It requires fewer read operations to a disk when the data and indexes are not scattered among many sections of the disk. (Data and indexes can get scattered as you update and add to the file.)

- Adjust parameters for input-output operations so that they best accommodate the size of the record

It is often better for input-output operations to transfer more than one record at a time. One record at a time is the DATATRIEVE default.

11.4 Restructuring a Domain

If you have not stored data in a file or do not want to keep data already stored, you can simply enter a new DEFINE FILE command to:

- Change file organization, storage options, or keys
- Create a file that incorporates changes made to a record definition

If you want to save records you have stored in a file, the commands and statements you enter depend on the changes you want to make.

11.4.1 Changing Only File Organization, Storage Options, and Keys

Example 11-2 illustrates the procedure you can follow when you want to make the following changes but do not want to lose data you have already stored:

- Change file organization from sequential to indexed or the reverse
- Add, delete, or change keys for an indexed file

- Reserve more storage space for future file expansion (ALLOCATION clause)
- Reserve maximum storage space for each variable length record (MAX)

Example 11-2 changes EMPLOYEES.DAT from an indexed file to a sequential one. Comment lines begin with an exclamation point (!) and explain the next input line.

Example 11-2: Restructuring a Domain to Change File Organization

```
DTR> ! Set up READ access to the original domain. Use an alias
DTR> ! to identify the relationship between the record definition and
DTR> ! the old file. (OLD is the alias in this example but you can
DTR> ! select another if you prefer.)
DTR> !
DTR> READY EMPLOYEES AS OLD
DTR> !
DTR> ! Create an empty file with the DEFINE FILE command of your
DTR> ! choice.
DTR> !
DTR> DEFINE FILE FOR EMPLOYEES
DTR> !
DTR> ! Set up WRITE access to your restructured domain. Use an alias
DTR> ! to identify the relationship between the record definition and
DTR> ! the new file. (NEW is the alias in this example but you can
DTR> ! select another if you prefer.)
DTR> !
DTR> READY EMPLOYEES AS NEW WRITE
DTR> !
DTR> ! Store records in the new file with a Restructure statement.
DTR> !
DTR> NEW = OLD
DTR> !
DTR> ! End access to NEW and OLD.
DTR> !
DTR> FINISH NEW, OLD
DTR> !
DTR> ! You can now ready the domain with its given name and can
DTR> ! access records in the new file.
DTR>
```

Before you start the restructure operation to create a new data file, find out the version number of the data file you are using for the domain. If you make a mistake or if the system fails in the middle of the restructure operation, you can delete all files with version numbers higher than this one and start the restructure operation again. The file the domain uses depends on how it is specified in the domain definition:

- If no version number is included on the file specification in the domain definition (usually it is not), then the domain uses the file of that name with the highest version number in the directory where it is stored. Exit DATATRIEVE and find out what this version number is and write it down.

- If a version number is included in the file specification in the domain definition, write down the version number. Restructuring a domain that contains a file specification with a version number involves a step not included in Example 11-2. *After* you ready OLD for READ access but *before* you define a new file, edit the domain definition to remove the version number from the file specification. Then continue with the DEFINE FILE command.

11.4.2 Changing the Fields Defined in the Record Definition

You can make some record definition changes without performing a restructure operation. You can:

- Add, change, or remove QUERY_HEADER and EDIT_STRING clauses
- Change field names or add, change, or remove QUERY_NAME clauses

If you have any procedures stored that use the old field or query names, remember to change these names in the procedures.

- Add DEFAULT or MISSING VALUE clauses

It is your responsibility, however, to make sure the value you specify agrees with any default values already stored in the file.

- Add group fields

You have to be careful when adding group fields if there are REDEFINES fields in the record. In this case, before you add group fields, you might want to review the rules that apply to the REDEFINES clause. Refer to the REDEFINES clause in *VAX DATATRIEVE Reference Manual* for more information.

You must restructure a domain if you want to do any of the following:

- Add new fields to the record.
- Change the order of the fields in the record.
- Increase the size of a field.
- Eliminate some fields from the record.

- Decrease the size of a field or change its data type.

If you decrease the size of a field or change the type of data it stores, the existing values in records for that field can be truncated or stored incorrectly. This is just a warning. You can still decrease field size if you:

- Plan to store new values for that field in all the records
- Intend to decrease the size a text field (a field with Xs or A's in the PIC clause) that has too many character positions for any values it needs to store

Example 11-3 illustrates the steps you follow to change the record definition describing data already stored. The comment lines start with an exclamation point (!) and explain the input line that follows. To put the example in context, assume you want to change the size of the ZIP field in EMPLOYEES_REC from 5 to 9 characters.

Example 11-3: Restructuring a Domain to Change the Record Definition

```
DTR> ! If NO EDIT_BACKUP is in effect during your DATATRIEVE
DTR> ! session (SHOW EDIT will tell you if it is), you should
DTR> ! enter the following command to ensure that the old version
DTR> ! of your record definition is not deleted. If something goes
DTR> ! wrong during the restructure operation, you will need to
DTR> ! use the old version again.
DTR> !
DTR> SET EDIT_BACKUP
DTR> !
DTR> ! Set up READ access to the original domain. Use an alias
DTR> ! to identify the relationship between the record definition and
DTR> ! the old file. (OLD is the alias in this example but you can
DTR> ! select another if you prefer.)
DTR> !
DTR> READY EMPLOYEES AS OLD
DTR> !
DTR> ! Edit the record definition. Do not change any field names.
DTR> ! If you do, DATATRIEVE will not be able to store the field
DTR> ! values. You can edit the record definition to change field
DTR> ! names after the restructure operation is completed.
DTR> !
DTR> EDIT EMPLOYEES_REC
```

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

```

DTR> !
DTR> !
DTR> ! Create an empty file with the DEFINE FILE command of your
DTR> ! choice.
DTR> !
DTR> DEFINE FILE FOR EMPLOYEES KEY = EMPLOYEE_ID
DTR> !
DTR> ! Set up WRITE access to the restructured domain. Use an alias
DTR> ! to identify the relationship between the record definition and
DTR> ! the new file. (NEW is the alias in this example but you can
DTR> ! select another if you prefer.)
DTR> !
DTR> READY EMPLOYEES AS NEW WRITE
DTR> !
DTR> ! Store records in the new file with a Restructure statement.
DTR> !
DTR> NEW = OLD
DTR> !
DTR> ! End access to OLD and NEW.
DTR> !
DTR> FINISH OLD, NEW
DTR> !
DTR> ! You can now ready the domain with its given name and
DTR> ! DATATRIEVE accesses records in the new file.
DTR>

```

Before you start the restructure operation to change a record definition, find out the version numbers of the data file and record definition you are using for the domain. If you make a mistake or if the system fails in the middle of the restructure operation, you can delete all files and record definitions with version numbers *higher* than the ones with which you started. You can then start the restructure operation again.

The data file the domain uses depends on how it is specified in the domain definition:

- If no version number is included in the file specification in the domain definition (usually it is not), then the domain uses the file of that name with the highest version number in the VMS directory where it is stored. Exit DATATRIEVE and find out what this version number is and write it down.
- If a version number is included in the file specification in the domain definition, write down that version number. Restructuring a domain that contains a file specification with a version number involves a step not included in Example 11-3. *After* you ready OLD for READ access but *before* you define a new file, edit the domain definition to remove the version number from the file specification. Then continue by editing the record definition and defining a new file.

The record definition your domain uses depends on how it is specified in the domain definition:

- If no version number is included on the record specified in the domain definition (usually it is not), then your domain uses the record definition of that name with the highest version number in the CDD directory where it is stored. Enter a `SHOW RECORDS` command to see how many versions of the record are in the directory. Write down the highest version number.
- If a version number is appended to the record in the domain definition, write down that version number. Restructuring a domain that specifies a record definition with a version number involves a step not included in Example 11-3. *After* you ready `OLD` for `READ` access but *before* you define a new file, edit the domain definition to remove the version number from the record. Then continue with the `EDIT` record-name command.

11.4.3 Restructuring a Domain to Add Its Records to Another Domain

Sometimes you might need to merge records from two domains that store the same data using different field names. In this case, you cannot simply ready the two domains and use a `Restructure` statement (`domain-name1 = domain-name2`) to store the records from one data file into the other. You must use the `FOR` statement and a `STORE` statement that explicitly stores each elementary field. If the two domains you are merging have the same names, you have to use an alias clause when you ready them.

In the following example, the domains have different names, so the alias clause is not necessary. All the records from `EMPLOYEES_BOSTON` are being stored into `EMPLOYEES_ALL`. In the `STORE` statement, the field names for `EMPLOYEES_ALL` are on the left of the equal sign (=) and the field names for `EMPLOYEES_BOSTON` are on the right:

```
DTR> READY EMPLOYEES_BOSTON
DTR> READY EMPLOYEES_ALL SHARED WRITE
DTR> FOR EMPLOYEES_BOSTON
CON> STORE EMPLOYEES_ALL USING
CON> BEGIN
CON>     EMPLOYEE_ID = EMP_ID
CON>     LAST_NAME = NAME_LAST
CON>     FIRST_NAME = NAME_FIRST
CON>     MIDDLE_INITIAL = INIT
CON>     .           .           .
CON>     .           .           .
CON>     .           .           .
CON> END
```

This operation uses statements that have not been discussed so far in this book. You can refer to Chapter 16 for more information about the `STORE` statement and to Chapter 17 for information about the `FOR` and `BEGIN-END` statements.

Defining Tables 12

This chapter tells you how to create and use dictionary tables and domain tables.

Both types of DATATRIEVE tables associate pairs of values. A dictionary table might pair zip codes with corresponding towns and states, for example. A domain table might associate employee identification numbers with employee names.

To save storage space, store the shorter of the two values in several domains in your database and store the longer value only in a dictionary table or in one domain that is the base for a domain table. You can access the longer values through the table with simple clauses that are easy to remember.

You can also validate field values by using a table. This table function is very useful when you need to store the same field in more than one domain. Using a table, you can make sure that the employee identification number for Ann Ducane, for example, is the same in all the places it is stored.

You create both types of DATATRIEVE tables with the DEFINE TABLE command. The syntax for the command differs, depending on the kind of table you want to create. The name you choose for a table definition cannot duplicate the name of any other object in the CDD directory where it is stored. Table names:

- Must begin with a letter
- Can consist only of letters, digits, hyphens, and underscores
- Must not duplicate a DATATRIEVE keyword
- Must not contain blanks
- Must be from 1- to 31- characters long
- Must end with a letter or digit

12.1 Creating Dictionary Tables

DATATRIEVE responds faster when you use a dictionary table than when you use a domain table. That is because the table definition itself specifies all the value pairs you want to access.

Example 12-1 creates the dictionary table DEPARTMENTS_TABLE. This table pairs department code values with corresponding department names. The comment lines in the example start with an exclamation point (!) and give you information about the next requirement or option in the command.

Example 12-1: Defining a Dictionary Table

```
DTR> ! Start your definition with the keywords DEFINE TABLE,
DTR> ! followed by the name you want for the table.
DTR> !
DTR> DEFINE TABLE DEPARTMENTS_TABLE
DFN> !
DFN> ! You can include the optional QUERY_HEADER clause to specify
DFN> ! a column header for table values when you display them.
DFN> !
DFN> QUERY_HEADER IS "DEPARTMENT"/"NAME"
DFN> !
DFN> ! You should include the optional EDIT_STRING clause to specify
DFN> ! how you want table values displayed. If you omit this clause
DFN> ! and do not include an edit string in a PRINT statement,
DFN> ! DATATRIEVE uses X(80) to display the values.
DFN> !
DFN> EDIT_STRING IS X(20)
DFN> !
DFN> ! Now enter the value pairs. The colon (:) is required to
DFN> ! separate values in the pair. Any spaces before and after
DFN> ! the colon are optional. You need the quotation marks to
DFN> ! preserve lowercase values or to enter values that are more
DFN> ! than one word (General Sales, for example).
DFN> !
DFN> "ADMN" : "Administration"
DFN> "ENG" : "Engineering"
DFN> "MKTG" : "Marketing"
DFN> "MNFG" : "Manufacturing"
DFN> "PERS" : "Personnel"
DFN> "SALE" : "General Sales"
DFN> !
DFN> ! The ELSE clause is optional. If you include it, DATATRIEVE
DFN> ! substitutes it for any values it finds in a domain and cannot
DFN> ! find in the table. If you omit it, DATATRIEVE displays an
DFN> ! error message when it cannot find the value in the table.
DFN> !
DFN> ELSE "Invalid Dept."
DFN> !
DFN> ! You must end your definition with the keyword END_TABLE.
DFN> !
DFN> END_TABLE
DTR>
```

As you define a dictionary table, DATATRIEVE checks for syntax errors. For example, if you enter a semicolon (;) in place of the required colon (:), DATATRIEVE prints an error message on your terminal and aborts the DEFINE TABLE command. To correct the error, type EDIT and press the RETURN key. (You cannot follow EDIT with the name of the table until your definition is stored.) Remember that while you are using the editor, DATATRIEVE does not check for syntax errors. If you get an error message when you exit the editor, you can immediately type EDIT and press the RETURN key to try again.

Use the SHOW command to display the names of tables stored at your current dictionary location and to display a table definition:

```
DTR> SHOW TABLES
Tables:
      DEPARTMENTS_TABLE;1          LOCATION_TABLE;3

DTR> SHOW DEPARTMENTS_TABLE
TABLE DEPARTMENTS_TABLE
QUERY_HEADER IS "DEPARTMENT"/"NAME"
EDIT_STRING IS X(20)
"ADMN" : "Administration"
"ENG"  : "Engineering"
"MKTG" : "Marketing"
"MNFG" : "Manufacturing"
"PERS" : "Personnel"
"SALE" : "General Sales"
ELSE "Invalid Dept."
END_TABLE

DTR> SHOW LOCATION_TABLE
TABLE LOCATION_TABLE
QUERY_HEADER "TOWN AND STATE"
EDIT_STRING X(25)
"02174" : "Arlington, MA"
"03051" : "Hudson, NH"
"03055" : "Milford, NH"
"03060" : "Nashua, NH"
"03061" : "Nashua, NH"
"07724" : "Eatontown, NJ"
ELSE "Fix table or record!"
END_TABLE
DTR>
```

12.2 Creating Domain Tables

A domain table definition does not contain all the value pairs you want to associate. It contains a pair of field names. The values for the fields are stored in the data files associated with domains. Usually several domains contain values for the shorter field and only one domain contains values for the longer field. In the sample personnel system used in this book, for example, several domains contain EMPLOYEE_ID values but only the EMPLOYEES domain contains EMPLOYEE_NAME values.

Example 12-2 defines the domain table WHO_IS_IT that associates EMPLOYEE_ID with EMPLOYEE_NAME. The comment lines in the example begin with an exclamation point (!) and provide information about the input that follows.

Example 12-2: Defining a Domain Table

```
DTR> ! Start your definition with DEFINE TABLE, followed by the
DTR> ! name of your table. Then enter FROM, followed by the name
DTR> ! of the domain containing both fields you want to relate.
DTR> !
DTR> DEFINE TABLE WHO_IS_IT FROM EMPLOYEES
DFN> !
DFN> ! You can include the optional QUERY_HEADER clause to specify
DFN> ! a column header for table values when you display them. If
DFN> ! the column header uses fewer character positions than the
DFN> ! associated table value, you can include spaces to position
DFN> ! the header where you want it.
DFN> !
DFN> QUERY_HEADER IS "          EMPLOYEE NAME          "
DFN> !
DFN> ! You should include the optional EDIT_STRING clause to specify
DFN> ! how you want table values displayed. If you omit this clause
DFN> ! and do not include an edit string in a PRINT statement,
DFN> ! DATATRIEVE uses X(80) to display the values.
DFN> !
DFN> EDIT_STRING IS X(36)
DFN> !
DFN> ! Now enter the field names. The colon (:) is required to
DFN> ! separate them. Any spaces before and after the colon are
DFN> ! optional. The keyword USING is also optional.
DFN> !
DFN> USING EMPLOYEE_ID : EMPLOYEE_NAME
DFN> !
DFN> ! The ELSE clause is optional. If you include it, DATATRIEVE
DFN> ! substitutes it for any values it finds in one domain and cannot
DFN> ! find in the other (table) domain. If you omit it, DATATRIEVE
DFN> ! displays an error message when it cannot find the value in the
DFN> ! domain on which the table is based. Use quotation marks to pre-
DFN> ! serve lowercase letters or if the ELSE value contains spaces.
DFN> !
DFN> ELSE "ID not in EMPLOYEES."
DFN> !
DFN> ! You must end your definition with the keyword END_TABLE.
DFN> !
DFN> END_TABLE
DTR>
```

12.3 Using DATATRIEVE Tables

You can reference tables in VALID IF clauses in record definitions and in any DATATRIEVE statement whose format allows it.

12.3.1 Access Privileges Needed to Use Tables

Table definitions, like all CDD objects created by DATATRIEVE, have associated ACLs that determine who can use them. The following information is important if you are creating tables to be used by people other than yourself or if you modify the ACLs for objects you create in your private branch of the CDD.

To use a dictionary table, users must have:

- P (PASS_THRU) privilege to the directory containing the table definition
- P (PASS_THRU), S (SEE), and E (EXECUTE/EXTEND) privileges to the table definition itself

To use a domain table, users must have:

- P (PASS_THRU) privilege to the directory containing the table definition
- P (PASS_THRU), S (SEE), and E (EXECUTE/EXTEND) privileges to the table definition itself
- P (PASS_THRU), S (SEE), and E (EXECUTE/EXTEND) privileges to the domain and record definitions on which the table is based
- P (PASS_THRU) privilege to the directory or directories containing the domain and record definitions on which the table is based

12.3.2 Accessing Values in Tables

You use the DATATRIEVE keywords IN, NOT IN, and VIA to access table values. The following example illustrates how these keywords work:

```
DTR> READY JOB_HISTORY
DTR> !
DTR> ! Create a collection of all the records in JOB_HISTORY that
DTR> ! have department codes in DEPARTMENTS_TABLE.
DTR> !
DTR> FIND JOB_HISTORY WITH DEPARTMENT_CODE IN DEPARTMENTS_TABLE
[194 records found]
DTR> !
DTR> ! Print values for two fields in the collection's records
DTR> ! and append a related field to each record by accessing the
DTR> ! table.
DTR> !
DTR> PRINT ALL EMPLOYEE_ID, DEPARTMENT_CODE, DEPARTMENT_CODE VIA
CON> DEPARTMENTS_TABLE
```

(continued on next page)

EMPLOYEE ID	DEPARTMENT CODE	DEPARTMENT NAME
00168	ENG	Engineering
00169	MNFG	Manufacturing
00171	ENG	Engineering
00172	SALE	General Sales
00173	MNFG	Manufacturing
00176	MNFG	Manufacturing
00182	PERS	Personnel
00183	PERS	Personnel
00184	MKTG	Marketing
00184	PERS	Personnel
00188	ADMN	Administration
00190	ADMN	Administration
00194	ENG	CTRL/C

^C

Execution terminated by operator.

```

DTR> ! Now form a collection of all records in JOB_HISTORY whose
DTR> ! department codes are not in the table.
DTR> !
DTR> FIND ALL JOB_HISTORY WITH DEPARTMENT_CODE NOT IN
CON> DEPARTMENTS_TABLE
[705 records found]
DTR> !
DTR> ! Print values for two fields in the collection's records
DTR> ! and append a related field to each record by accessing the
DTR> ! table. Note that the value in the ELSE clause of the table
DTR> ! definition appears.
DTR> !
DTR> PRINT ALL EMPLOYEE_ID, DEPARTMENT_CODE, DEPARTMENT_CODE VIA
CON> DEPARTMENTS_TABLE

```

EMPLOYEE ID	DEPARTMENT CODE	DEPARTMENT NAME
00164	MBMN	Invalid Dept.
00164	MCBM	Invalid Dept.
00165	ELGS	Invalid Dept.
00165	PHRN	Invalid Dept.
00165	MBMF	Invalid Dept.
00165	MTEL	Invalid Dept.
00166	PRMG	Invalid Dept.
00166	MB	CTRL/C

^C

Execution terminated by operator.

```

DTR> !
DTR> ! Assume these values are valid department codes that you
DTR> ! need to add to the table. By reducing the current collection
DTR> ! to unique department code values and then printing the
DTR> ! reduced collection, you get a list of what needs to be
DTR> ! added to DEPARTMENTS_TABLE.
DTR> !
DTR> REDUCE TO DEPARTMENT_CODE
DTR> PRINT ALL

```

DEPARTMENT
CODE

ELEL
ELGS
ELMC
MBMF
MBMN
MBMS
MCBM
MCBS
MGVT
MSCI
MSMG
MTEL
PERL
PHRN
PRMG
SEUR
SUNE
SUSA
SUSO
SUWE

DTR>

See the next section on how to edit a table definition. A later section explains how to use the corrected table to prevent users from entering invalid values when storing or modifying records.

12.3.3 Editing Table Definitions

You edit your table definition by entering `EDIT` followed by the table name. Your table definition is copied to an editing buffer where you can make the changes you want.

When you access a table, it is loaded into your `DATATRIEVE` workspace where it remains until you either remove it or exit the session. The `SHOW READY` command displays the names of any tables currently in your workspace.

If the table is loaded into your `DATATRIEVE` workspace at the time you edit the definition, you must remove the table from your workspace and access it again before the changes you made can take effect. To remove a table from your workspace, enter `RELEASE` followed by the table name.

12.3.4 Validating Values with Tables

By referring to a dictionary or domain table in a VALID IF clause in a record definition, you can validate data entered for a field before it is stored in a record:

```
DTR> SHOW JOB_HISTORY_REC
RECORD JOB_HISTORY_REC USING
01 JOB_HISTORY_REC.
    05 DEPARTMENT_CODE PIC X(4)
        VALID IF DEPARTMENT_CODE IN
        DEPARTMENTS_TABLE.
    05 EMPLOYEE_ID PIC X(5)
        VALID IF EMPLOYEE_ID IN
        WHO_IS_IT.
    05 JOB_CODE PIC X(4).
    05 JOB_START USAGE DATE.
    05 JOB_END USAGE DATE.
    05 REVIEW_DATE USAGE DATE.
    05 SUPERVISOR_ID PIC X(5).
```

```
DTR> ! When users store or modify records in the JOB_HISTORY domain,
DTR> ! DATATRIEVE checks a value entered for DEPARTMENT_CODE against
DTR> ! codes in DEPARTMENTS_TABLE and a value entered for EMPLOYEE_ID
DTR> ! against codes in WHO_IS_IT.
DTR> !
```

```
DTR> READY JOB_HISTORY WRITE
DTR> STORE JOB_HISTORY
Enter DEPARTMENT_CODE: XXXX
Validation error for field DEPARTMENT_CODE.
Re-enter DEPARTMENT_CODE: SALE
Enter EMPLOYEE_ID: 00000
Validation error for field EMPLOYEE_ID.
Re-enter EMPLOYEE_ID: 00168
Enter JOB_CODE:
```

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

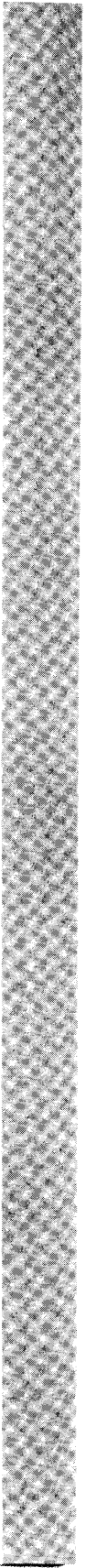
```
DTR>
```

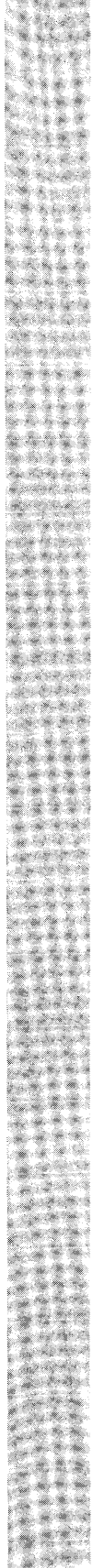
12.4 Choosing Between Dictionary and Domain Tables

To decide which type of table to use, keep the following guidelines in mind:

- Dictionary tables lend themselves to interactive updates. You add or change entries to the table directly by editing the table definition. In addition, DATATRIEVE works faster with dictionary tables than with domain tables.
- Because a domain table does not contain the pairs of values it relates, it is automatically updated when the associated fields of its domain are changed. For applications where the values associated by a table change often, a domain table can be easier to maintain. Any statement that changes or adds records to the domain that is the basis for the table will also update the table itself.

Part IV
Data Retrieval and Maintenance





Starting and Ending Access to Data **13**

This chapter explains data access when you are using domains associated with VAX RMS (Record Management Services) data files.

Note

If you are accessing data managed by VAX DBMS or any of the VAX relational database products, refer to the chapters on those products in the *VAX DATATRIEVE User's Guide*. If you are accessing data stored on remote systems, refer to the chapter in the same manual on defining and accessing distributed domains. In these cases, there are **READY** command options and access default differences that are not discussed in this book.

You access data with a **READY** command, which contains the name of the domain or domains associated with the records you want to see. The **SHOW READY** command tells you what domains are currently readied and what options were selected for their use. You end access to data with a **FINISH** command or by exiting your **DATATRIEVE** session.

Example 13-1 shows some sample **READY** and **FINISH** commands for a **DATATRIEVE** session. For Example 13-1, assume the user needs to store data for a new employee but first needs to check information stored in the **JOBS** domain and **JOBS_TABLE** table.

Example 13-1: Starting and Ending Access to Data

DTR) SHOW DOMAINS

Domains:

COLLEGES;1 DEGREES;1 EMPLOYEES;1 JOBS;1
JOB_HISTORY;1 SALARY_HISTORY;1

DTR) READY JOBS

DTR) SHOW FIELDS FOR JOBS

JOBS

 JOB

 JOB_CODE (Character string, primary key)
 MINIMUM_SALARY (Number)
 MAXIMUM_SALARY (Number)
 WAGE_CLASS (Character string)

DTR) PRINT JOB_CODE, JOB_CODE VIA JOBS_TABLE, MINIMUM_SALARY,
CON) MAXIMUM_SALARY, WAGE_CLASS OF JOBS WITH JOB_CODE CONTAINING
CON) "GM"

JOB CODE	JOB TITLE	MINIMUM SALARY	MAXIMUM SALARY	WAGE CLASS
APGM	Associate Programmer	\$15,000.00	\$24,000.00	4
PRGM	Programmer	\$20,000.00	\$35,000.00	4
SPGM	Systems Programmer	\$25,000.00	\$50,000.00	4

DTR) READY EMPLOYEES SHARED WRITE, JOB_HISTORY SHARED WRITE,

CON) SALARY_HISTORY SHARED WRITE

DTR) SHOW READY

Ready sources:

 SALARY_HISTORY: Domain, RMS indexed, shared write
 (CDD\$TOP.PERSONNEL.SALARY_HISTORY;1)
 JOB_HISTORY: Domain, RMS indexed, shared write
 (CDD\$TOP.PERSONNEL.JOB_HISTORY;1)
 EMPLOYEES: Domain, RMS indexed, shared write
 (CDD\$TOP.PERSONNEL.EMPLOYEES;1)
 JOBS: Domain, RMS indexed, protected read
 (CDD\$TOP.PERSONNEL.JOBS;1)

Loaded tables:

 JOBS_TABLE: Dictionary table
 (CDD\$TOP.PERSONNEL.JOBS_TABLE;2)

DTR) FINISH JOBS

DTR) RELEASE JOBS_TABLE

DTR) SHOW READY

Ready sources:

 SALARY_HISTORY: Domain, RMS indexed, shared write
 (CDD\$TOP.PERSONNEL.SALARY_HISTORY;1)
 JOB_HISTORY: Domain, RMS indexed, shared write
 (CDD\$TOP.PERSONNEL.JOB_HISTORY;1)
 EMPLOYEES: Domain, RMS indexed, shared write
 (CDD\$TOP.PERSONNEL.EMPLOYEES;1)

No loaded tables.

```
DTR> STORE EMPLOYEES; STORE JOB_HISTORY; STORE SALARY_HISTORY
```

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

```
DTR> FINISH
```

```
DTR> SHOW READY
```

```
No ready sources.
```

```
No loaded tables.
```

```
DTR>
```

13.1 Readying Domains

When you ready a domain, DATATRIEVE loads the record definition associated with the domain into your workspace and opens the associated data file. In addition to the domain name, a READY command can include:

- An alternate name (alias) for the domain while you are using it
- The access options other users have to a domain while you are using it (PROTECTED, SHARED, or EXCLUSIVE)
- The access mode you need for the operation you want to perform (READ, WRITE, MODIFY, or EXTEND)

You have to specify an alias only if two domains with the same given name will be ready at the same time. This situation can occur when you are accessing domains stored in different dictionary directories and when you are restructuring your database. (In the command READY EMPLOYEES AS NEW WRITE, NEW is an alias.) When you ready a domain under an alias, you must use the alias rather than the domain's given name in any subsequent statements or commands during that session. DATATRIEVE does not recognize the readied domain if you use the given name.

PROTECTED is the access option that applies if you do not specify one.

PROTECTED means that if other users ready the domain while you are using it, they can use it only to retrieve and display data. They cannot modify, erase, or add records.

READ is the access mode that applies if you do not specify one. READ means that you can use the domain only to retrieve and display data. You cannot store, erase, or modify records without readying the domain again to specify a new access mode.

Tables 13-1, 13-2, and 13-3 summarize information about access options and modes that affect you and other people using the domain.

Table 13-1: Access Options

Option	Access Constraints
PROTECTED	Any other user can have only READ access to records in the domain or relation. No other user can have WRITE, MODIFY, or EXTEND access to the records in the domain or relation. This option is the default for domains containing records from RMS files and for all view domains.
SHARED	Any other user can have access to the domain or relation at the same time in any access mode.
EXCLUSIVE	No other user can have access to the domain or relation at the same time in any access mode.

Table 13-2: Access Modes

Mode	Type of Access	Privileges Needed
READ	You can only retrieve records. (Default)	P (PASS_THRU), S (SEE), and R (DTR_READ), M (DTR_MODIFY), or W (DTR_WRITE)
MODIFY	You can retrieve and modify records.	P (PASS_THRU), S (SEE), and M (DTR_MODIFY) or W (DTR_WRITE)
WRITE	You can retrieve, modify, store, and erase records.	P (PASS_THRU), S (SEE), and W (DTR_WRITE)
EXTEND	You can only store records.	P (PASS_THRU), S (SEE), and W (DTR_WRITE) or E (DTR_EXTEND/EXECUTE)

In addition to the domain definition privileges listed in Table 13-2, you must also have P (PASS_THRU), S (SEE), and E (EXTEND/EXECUTE) privileges for the associated record definition to ready a domain for any operation.

Table 13-3: Multiuser Access

You Ready a Domain	Another User Can Then Ready the Domain	Your Effect on Other Users	Other Users' Effect on You
EXCLUSIVE READ EXCLUSIVE WRITE	No access	No one else can read the file.	No effect.
PROTECTED READ	PROTECTED READ SHARED READ	No one else can write to the file.	No effect.
PROTECTED WRITE	SHARED READ	No one else can write to the file.	No effect.
SHARED READ	PROTECTED READ PROTECTED WRITE SHARED READ SHARED WRITE	No user with WRITE access can select your selected record.	Users with WRITE access may change records you are reading or have read.
SHARED WRITE	SHARED READ SHARED WRITE	No one else can modify your selected record or the target record of your MODIFY or ERASE statement. You can modify a record another user has just modified.	You cannot write to the selected record of any other user. You cannot write to the target record of a MODIFY or ERASE statement entered by a SHARED WRITE user. A SHARED WRITE user can also write to a record you have just modified.

13.1.1 Defining Your Own Default Access

DATATRIEVE provides the following access options by default, depending on the source you are readying, if you do not supply an access option on the READY command line:

- PROTECTED (RMS sources)
- SNAPSHOT (Relational sources)
- SHARED (DBMS sources)

If you want to define your own default access option, use the logical name `DTR$READY_MODE`.

`DATATRIEVE` checks the definition of `DTR$READY_MODE` only when an access option is not found on the `READY` command line. You can assign a default to `DTR$READY_MODE` as follows:

- Use the `DATATRIEVE` function `FN$CREATE_LOG`. The following example changes the `READY` access of the RMS domain `YACHTS` from its default, `PROTECTED`, to `SHARED` access:

```
DTR> FN$CREATE_LOG ("DTR$READY_MODE", "SHARED")
DTR> FINISH
DTR> READY YACHTS
DTR> SHOW READY
Ready sources:
  YACHTS: Domain, RMS indexed, shared read
          (<_CDD$TOP.DTR32.DAB.YACHTS;3)
```

- Use either the `DCL ASSIGN` or `DEFINE` command as follows:

```
$ ASSIGN "SHARED" DTR$READY_MODE
$ DEFINE DTR$READY_MODE "SHARED"
```

- Use a combination of a synonym and a logical assignment:


```
DTR> DECLARE SYNONYM EXCL FOR EXCLUSIVE
DTR> FN$CREATE_LOG ("DTR$READY_MODE", "EXCL")
DTR> READY YACHTS
DTR> SHOW READY
Ready sources:
  YACHTS: Domain, RMS indexed, exclusive read
          (<_CDD$TOP.DTR32.DAB.YACHTS;3)
```

If you define `DTR$READY_MODE` using `FN$CREATE_LOG`, the definition lasts only until the end of the `DATATRIEVE` session. You can change the definition during the session, however.

See the *VAX DATATRIEVE Reference Manual* for information about access option error handling.

13.2 Finishing Domains

Use the `FINISH` command to end your control over one or more domains. If you specify more than one domain name in the `FINISH` command, enter commas to separate the domain names. If you enter the keyword `FINISH` by itself, or if you enter `FINISH ALL`, you end your control over all the domains you have readied.



Finishing domains is especially important if you have readied any domains with the **PROTECTED** or **EXCLUSIVE** access option and other users access those domains. The **PROTECTED** option keeps other users from updating the data file. The **EXCLUSIVE** access option locks out other users entirely. In addition, if you ready a domain with the **EXCLUSIVE** access option and that domain is the base for a domain table, you must finish the domain before you can use the domain table.





Retrieving Data the Easy Way: With Collections **14**

A **DATATRIEVE collection** is a group of records you gather from one or more sources with a **FIND** statement. Usually, record sources are readied domains. A collection stays in your workspace until it is superseded by another collection or until you remove it.

DATATRIEVE considers that many statements apply to a record selected from a collection unless you say otherwise. There are also some statements that apply only to collections. Working with collections, therefore, usually means that your statements can be simpler and shorter. You do not always have to tell **DATATRIEVE** where to look for data and can focus on what you want to do with it.

Example 14-1 illustrates creating a collection and some of the things you can do with it. The comment lines in the example begin with an exclamation point (!) and prepare you for the input that follows.

Example 14-1: Creating and Using a Collection

```
DTR> ! To find out how many employees have been willing to commute
DTR> ! from Massachusetts, create a collection of employee
DTR> ! records with MA listed as the state.
DTR> !
DTR> READY EMPLOYEES
DTR> FIND EMPLOYEES WITH STATE = "MA"
[36 records found]
DTR> !
DTR> ! DATATRIEVE groups these records in a collection named
DTR> ! CURRENT.
DTR> !
DTR> SHOW COLLECTIONS
Collections:
    CURRENT
```

(continued on next page)

DTR> SHOW CURRENT
 Collection CURRENT
 Domain: EMPLOYEES
 Number of Records: 36
 No Selected Record

DTR> !
 DTR> ! You can type PRINT ALL to display the records in CURRENT.
 DTR> !
 DTR> !
 DTR> PRINT ALL

ID	LAST NAME	FIRST NAME INIT	ADDRESS DATA	ZIP	SEX	SOCIAL SECURITY
00174	Myotte	Daniel V	95 Princeton Rd. 1/17/48	MA 03442	M	246 68 2816
00175	Siciliano	George	109 Old New Boston Rd. 5/25/41	MA 03835	M	136 17 0800
00191	Pfeiffer	Karen I	143 Hudson Rd. Marlborough			
.
.
.

DTR> !
 DTR> ! You do not need all the information in the record. You can use
 DTR> ! the REDUCE statement to specify a combination of field
 DTR> ! values that makes each record unique and to eliminate fields
 DTR> ! you are not interested in. (You would include ID if you
 DTR> ! suspect there might be employees with the same name living
 DTR> ! in the same town.)
 DTR> !
 DTR> REDUCE TO NAME, TOWN, STATE
 DTR> PRINT ALL

LAST NAME	FIRST NAME INIT	TOWN	STATE
Babbin	Cora	Boston	MA
Boutin	Janis S	Bennington	MA
Carmody	Susan	Marlborough	MA
Chandler	Christine E	Bennington	MA
Clarke	Aruwa Q	Cambridge	MA
.	.	.	.
.	.	.	.
.	.	.	.

```

DTR> !
DTR> ! Order the records in the collection according to
DTR> ! town.
DTR> !
DTR> SORT BY TOWN
DTR> !
DTR> ! Display the reordered records so only town and name
DTR> ! print on your screen.
DTR> !
DTR> PRINT ALL TOWN, NAME

```

TOWN	LAST NAME	FIRST NAME	INIT
Bennington	Delano	Al	F
Bennington	Comstock	Frederick	E
Bennington	Mistretta	Kathleen	G
Bennington	Rodrigo	Lisa	
Bennington	Turner	Alan	
Bennington	Lynch	Mary	F
Bennington	Chandler	Christine	E
Bennington	Boutin	Janis	S
Bennington	Rothwell	Dean	
Bennington	Myotte	Daniel	V
Bennington	Siciliano	Jesse	W
Boston	Harrison	Lisa	
Boston	Staples	Jerry	Z
Boston	Roberts	Joseph	V
.	.	.	.
.	.	.	.
.	.	.	.

```

DTR> !
DTR> ! Create a new collection from the CURRENT collection.
DTR> ! Make it contain only names from Boston.
DTR> !
DTR> FIND CURRENT WITH TOWN CONTAINING "BOSTON"
[6 records found]
DTR> PRINT ALL

```

LAST NAME	FIRST NAME	INIT	TOWN	STATE
Harrison	Lisa		Boston	MA
Staples	Jerry	Z	Boston	MA
Roberts	Joseph	V	Boston	MA
Iosca	Karen		Boston	MA
Babbin	Cora		Boston	MA
Sprout	Wayne		Boston	MA

```
DTR>
```


14.1 Specifying the Records You Want in a Collection

Example 14-1 uses the statement `FIND EMPLOYEES WITH STATE = "MA"` to form a collection. `EMPLOYEES WITH STATE = "MA"` is a record selection expression (RSE). The simplest RSE specifies only a record source. `FIND EMPLOYEES`, for example, creates a collection that contains all the records in the `EMPLOYEES` domain, rather than only those that specify `MA` in the `STATE` field.

There are six options you can include in an RSE to specify the records you want. When you include more than one of these options in an RSE, you must specify them in the order they appear in the following list. A simple `FIND` statement example illustrates each option:

- Record number restriction

```
DTR> FIND FIRST 5 EMPLOYEES
```

- A name for the group of records from each record source

```
DTR> FIND A IN EMPLOYEES
```

- A match of records from more than one source

```
DTR> FIND EMPLOYEES CROSS JOB_HISTORY OVER EMPLOYEE_ID
```

- Record contents restriction

```
DTR> FIND JOB_HISTORY WITH JOB_END MISSING
```

- Field restriction

```
DTR> FIND SALARY_HISTORY REDUCED TO EMPLOYEE_ID,  
DEPARTMENT_CODE, JOB_CODE
```

- Record order

```
DTR> FIND EMPLOYEES SORTED BY LAST_NAME
```

The following `FIND` statement shows you the order of these options when all of them appear in the same RSE. Using data from the `EMPLOYEE` and `JOB_HISTORY` domains, the statement creates a collection that contains current job information for ten employees:

```
DTR> FIND FIRST 10 A IN EMPLOYEES CROSS  
CON> B IN JOB_HISTORY OVER  
CON> EMPLOYEE_ID WITH JOB_END MISSING REDUCED TO  
CON> EMPLOYEE_ID, LAST_NAME, DEPARTMENT_CODE,  
CON> JOB_CODE SORTED BY DEPARTMENT_CODE, LAST_NAME
```

One of the nicest things about collections is you do not have to enter statements like that one. You can start out with a collection that contains more records than you need. You can then order the records and fields, eliminate fields, or drop records out of the collection in separate steps until you get exactly the data you want.

When reading this chapter, you will encounter expressions that include keywords such as =, CONTAINING, GT, LT, and MISSING. You can read the sections on using Boolean and value expressions in Chapter 18 to find out all the options DATATRIEVE provides for data retrieval. For now, here are a few guidelines and options to get you started:

- Enclosing values in quotation marks

The SHOW FIELDS command displays the name and description for all fields in the domains you have readied. If the field name is followed by “character string” or “date,” you put quotation marks around values for the field. If the field name is followed by “number,” then you do not put quotation marks around field values.

- Choosing keywords such as = (equals), GT (greater than), LT (less than)

These keywords (called **operators**) relate two value expressions. Most commonly, you precede an operator with a field name and follow it with a value appropriate for that field.

When you are using these operators and you specify values for fields described as character string, you must enter each letter using the case in which the letter is stored. For example, if you enter “TERRY” and the value for which you are searching was stored as “Terry”, DATATRIEVE will not find it. Similarly, you must distinguish underscores (_) from hyphens (-).

- Choosing the keyword CONTAINING (abbreviated CONT)

CONTAINING works more slowly than the other operators do when you specify values for fields that are keys to an indexed file, so use it sparingly. CONTAINING is not case-sensitive. You can use it to find character-string values when you are not sure what case was used when values were stored in a field.

The following sections explain in more detail how you create and work with collections.

14.2 Forming and Naming Collections

You must ready a domain (using any access mode other than EXTEND) before you can form a collection from it.

When the FIND statement executes, DATATRIEVE creates a collection consisting of the records specified in the RSE and names the collection CURRENT. Unless the FIND statement executes inside a procedure, DATATRIEVE also tells you how many records it found.

If you specify a name for the group of records in the FIND statement RSE, the collection has two names: CURRENT and the name you specify. You can refer to the collection by the name CURRENT or by the name you specify:

```
DTR> READY SALARY_HISTORY
DTR> FIND BIG_WIGS IN SALARY_HISTORY WITH
CON> SALARY_END MISSING AND SALARY_AMOUNT GT 50000
[38 records found]
DTR> SHOW CURRENT
Collection BIG_WIGS
  Domain: SALARY_HISTORY
  Number of Records: 38
  No Selected Record

DTR> SHOW COLLECTIONS
Collections:
  BIG_WIGS      (CURRENT)
```

When you use a FIND statement to form another collection, the new collection becomes the CURRENT collection. You can refer to the old collection only by the name you gave it. If you did not name the old collection, DATATRIEVE deletes it when the new one is formed:

```
DTR> FIND DRONES IN SALARY_HISTORY WITH
CON> SALARY_END MISSING AND SALARY_AMOUNT LT 50000
[281 records found]
DTR> SHOW CURRENT
Collection DRONES
  Domain: SALARY_HISTORY
  Number of Records: 281
  No Selected Record

DTR> SHOW COLLECTIONS
Collections:
  DRONES      (CURRENT)
  BIG_WIGS
```

DTR> PRINT

No record selected, printing whole collection.

EMPLOYEE ID	SALARY AMOUNT	SALARY START	SALARY END	REVIEW CODE
00165	\$11,676.00	1-Jul-1982		3
00166	\$18,497.00	7-Aug-1982		2
00167	\$17,510.00	21-Aug-1982		2
	.	.	.	
	.	.	.	
00359	\$93,340.00	18-Dec-1982		1

DTR> PRINT BIG_WIGS

EMPLOYEE ID	SALARY AMOUNT	SALARY START	SALARY END	REVIEW CODE
00164	\$51,712.00	14-Jan-1983		1
00172	\$55,413.00	17-Dec-1982		1
00195	\$51,586.00	10-Mar-1982		1
00200	\$51,019.00	21-Nov-1982		1
00204	\$87,143.00	21-Sep-1982		1
00215	\$55, CTRL/C			

^C

Execution terminated by operator.

DTR>

14.3 Choosing a Target Record for an Operation

You can use the SELECT statement to establish a target record for an operation. This is very useful when you want to erase or modify one or a few records from a data file. When you establish a selected record, you can simply type PRINT, ERASE, or MODIFY, and DATATRIEVE will know you are referring to the selected record. If you want to do something to the whole collection rather than the selected record, include the keyword ALL (PRINT ALL, MODIFY ALL, or ERASE ALL, for example):

DTR> READY EMPLOYEES MODIFY

DTR> FIND EMPLOYEES WITH LAST_NAME CONTAINING "SMITH"

[2 records found]

DTR> PRINT ALL LAST_NAME, ID, STREET, TOWN, STATE

LAST NAME	ID	STREET	TOWN	STATE
Smith	00165	120 Tenby Dr.	Chocorua	NH
Smith	00209	163 Lowell Rd.	Bristol	NH

(continued on next page)

```

DTR> SELECT
DTR> PRINT LAST_NAME, ID

  LAST NAME      ID
Smith            00165

DTR> MODIFY LAST_NAME
Enter LAST_NAME: Overton
DTR> PRINT LAST_NAME, ID

  LAST NAME      ID
Overton          00165

DTR> PRINT ALL LAST_NAME, ID

  LAST NAME      ID
Overton          00165
Smith            00209

DTR>

```

The example illustrates only two of the eight options you can use with a `SELECT` statement. Here is a complete list of `SELECT` options with an example for each. You can:

- Select the first record in the collection

`SELECT FIRST`

- Select the record in the collection positioned immediately after the current selected record

`SELECT NEXT`

This is the default. When you simply type `SELECT`, `DATATRIEVE` selects the next record in the collection. As the example in the beginning of this section illustrates, if you have not established a selected record, `DATATRIEVE` selects the first record in the collection when you type `SELECT` or `SELECT NEXT`.

- Select the record in the collection positioned immediately before the current selected record

`SELECT PRIOR`

- Select the last record in the collection

`SELECT LAST`

- Select the record whose ordinal position you specify

```
SELECT 5
```

The example specifies the fifth record in the collection. You can use an expression in place of an integer as long as the expression resolves to an integer value (COUNTER_FIELD + 1, for example, where COUNTER_FIELD contains 0 or an integer).

- “Unselect” a record for a collection

```
SELECT NONE
```

This option releases your control over your current selected record so that other users can access it.

- Name the collection from which you want to select (or unselect) a record

```
SELECT 2 MY_COLLECTION
```

The example specifies the second record in the collection named MY_COLLECTION. If you do not name a collection in a SELECT statement, DATATRIEVE selects the record from the CURRENT collection.

- Specify a WITH clause

```
SELECT FIRST MY_COLLECTION WITH LAST_NAME = "SMITH"
```

The example specifies the first record in MY_COLLECTION that has SMITH in the LAST_NAME field.

You can establish more than one selected record but only one for each collection. When you enter a statement that applies to a selected record, DATATRIEVE carries out the requested operation on the record you most recently selected. If your statement cannot apply to the most-recently selected record, DATATRIEVE tries to carry out the operation on a selected record for another collection. DATATRIEVE continues to check selected records you have available, in reverse order of their selection, until it can either execute the statement or determine an error condition.

When you are beginning to work with DATATRIEVE, this rather elaborate description of how DATATRIEVE works probably is not important to you. Generally, beginners work with one collection at a time and do not have more than one selected record available. As you gain experience, however, you may be working with more than one collection and have several selected records established. You might also be working with a DATATRIEVE procedure written by an experienced user who manipulates multiple selected records. In this case, you need to know more about how DATATRIEVE processes your input. The appendix on name recognition and record context in the *VAX DATATRIEVE User's Guide* provides more detailed information on this topic.

14.4 Restricting Record Fields to the Ones You Need

As an alternative to putting a REDUCED TO clause in the FIND statement RSE, you can use a REDUCE statement to keep only the fields you want to work with in a collection:

```
DTR> READY EMPLOYEES
DTR> FIND EMPLOYEES WITH SEX = "F"
[105 records found]
DTR> PRINT ALL
```

ID	LAST NAME	FIRST NAME INIT	ADDRESS DATA	ZIP	SEX	SOCIAL SECURITY
00167	Kilpatrick	Janet				
	143 Pine St.	Marlow	NH	03456	F	889 84 0384
	3/05/37					
00169	Gray	Susan 0				
	51 Maple St.	Etna	NH	03750	F	475 94 2624
	8/13/38					
00171	D'Amico	Aruwa				
	67 Underhill St.	Sandown	NH	03873	F	185 77 9984
	1/29/32					
00172	Peters	Janis K				
	13 Mobile C					

Execution terminated by operator

```
DTR> ! You can control record display by specifying fields in the
DTR> ! PRINT statement. This does not change the records in the
DTR> ! collection.
DTR> !
DTR> PRINT ALL EMPLOYEE_ID, NAME, STATE
```

ID	LAST NAME	FIRST NAME INIT	STATE
00167	Kilpatrick	Janet	NH
00169	Gray	Susan 0	NH
00171	D'Amico	Aruwa	NH
00172	Peters	Janis K	NH
00179	Vermouth	Meg	NH
00185	Stadecker	Hope E	NH
00186	Watters	Cora	NH
00188	Clarke	Karen G	NH
00191	Pfeiffer	Karen I	MA
00192	Connolly	Christine	NH
00194	Morrison	Mary Lou U	NH
00196	Clarke	Mary	NH
00197			

Execution terminated by operator.

```
DTR> PRINT ALL
```

ID	LAST NAME	FIRST NAME INIT	ADDRESS DATA	ZIP	SEX	SOCIAL SECURITY
00167	Kilpatrick	Janet				
	143 Pine St.	Marlow	NH	03456	F	889 84 0384
	3/05/37					
00169	Gray	Susan				
	51 Maple St.	Etna	NH	03750	F	475 94 2624
	8/13/38					

^C
Execution terminated by operator.

DTR> !
DTR> ! The REDUCE statement changes the records in the collection
DTR> ! to unique combinations of the fields you specify.
DTR> ! Duplicate values, if any exist, no longer appear in the
DTR> ! collection. The first REDUCE statement that follows does not
DTR> ! change the number of records in the collection. Its purpose
DTR> ! is to reduce the number of fields each record contains. The
DTR> ! second REDUCE statement that follows does change the number
DTR> ! of records in the collection. It illustrates the power of
DTR> ! reducing a collection to unique values.

DTR> !
DTR> REDUCE TO EMPLOYEE_ID, NAME, STATE
DTR> PRINT ALL

ID	LAST NAME	FIRST NAME INIT	STATE
00167	Kilpatrick	Janet	NH
00169	Gray	Susan	0 NH
00171	D'Amico	Aruwa	NH
00172	Peters	Janis	K NH
00179	Vermouth	Meg	NH
00185	Stadecker	Hope	E NH

00 CTRL/C
^C
Execution terminated by operator.

DTR> REDUCE TO STATE
DTR> PRINT ALL

STATE

MA
NH

DTR>

The REDUCE statement comes in handy when your collection data results from crossing records from two or more domains. A cross operation produces records with one or more duplicate fields. You can use the REDUCE statement to make sure that all fields are unique. (See a later section in this chapter for more information.)

Note

If a record contains between 80 and 132 characters, you can do the following so that it displays on one line:

- Type FN\$WIDTH(132) and press the RETURN key.
- Type SET COLUMNS_PAGE=132 and press the RETURN key.

This changes system, terminal, and DATATRIEVE defaults for screen width setting so that up to 132 characters are displayed on each line. To set your screen width back to 80 characters per line, enter the same commands but substitute 80 for 132.

14.5 Sorting Records

As an alternative to putting a SORTED BY clause in a FIND or PRINT statement RSE, you can use the SORT statement to put collection records in the order you want:

```
DTR> ! Assume you want to find out the employee distribution within
DTR> ! job codes in each department.
DTR> !
DTR> READY JOB_HISTORY
DTR> FIND JOB_HISTORY WITH JOB-END MISSING
[338 records found]
DTR> PRINT ALL
```

EMPLOYEE ID	JOB CODE	JOB START	JOB END	DEPARTMENT CODE	SUPERVISOR ID	REVIEW DATE
00164	DMGR	21-Sep-1981		MBMN	00359	14-Jul-1983
00165	DGFR	8-Mar-1981		MBMF	00358	1-Jan-1983
00166	APGM	12-Aug-1981		MBMS	00229	7-Feb-1983
00167	APGM	26-Aug-1981		MBMN	00359	21-Feb-1983
00168	SPGM	18-Feb-1982		MGVT	00267	15-Jun-1983
00169	SPGM	28-Mar-1981		SUNE	00354	17-Jul-1983
00170	SCTR	26-Nov-1980				

```
^C
Execution terminated by operator.
```

```
DTR> SORT BY DEPARTMENT_CODE, JOB_CODE
DTR> PRINT ALL
```

EMPLOYEE ID	JOB CODE	JOB START	JOB END	DEPARTMENT CODE	SUPERVISOR ID	REVIEW DATE
00472	DSUP	27-Apr-1981		ADMN	00225	19-Apr-1983
00300	EENG	11-Feb-1982		ADMN	00225	11-Aug-1982
00188	EENG	8-Apr-1982		ADMN	00225	2-Aug-1983
00330	JNTR	6-Feb-1981		ADMN	00225	1-Dec-1982
00438	MENG	25-Apr-1980		ADMN	00225	14-Aug-1982
00190	MENG	25-Feb-1982		ADMN	00225	22 CTRL/C

^C

Execution terminated by operator.

DTR> !
DTR> ! Usually, you want your records displayed according to the way
DTR> ! you sorted them. The following PRINT statement does this. It
DTR> ! also retrieves the name for each employee from the domain table
DTR> ! WHO_IS_IT.
DTR> !
DTR> PRINT ALL DEPARTMENT_CODE, JOB_CODE, EMPLOYEE_ID,
CON> EMPLOYEE_ID VIA WHO_IS_IT

DEPARTMENT CODE	JOB CODE	EMPLOYEE ID	EMPLOYEE NAME	
ADMN	DSUP	00472	Delano	Al F
ADMN	EENG	00300	Gramby	Marjorie
ADMN	EENG	00188	Clarke	Karen G
ADMN	JNTR	00330	Williams	Christine B
ADMN	VPSD	00415	Mistretta	Kathleen G
ELEL	APGM	00377	Lobdell	Lawrence V
ELEL	EENG	00238	Flynn	Peter
ELEL	EENG	00428	Augusta	Thomas
ELEL	GFER	00231	Clairmont	Rick
ELEL	GFER	00240	Johnson	Bill R
ELEL	GFER	00461	Boutin	George
ELEL	GFER	00222	Lasch	Norman

DTR>

If you specify more than one field by which you want to sort the records, remember to always include a comma to separate the fields in your list.

14.6 Forming a Collection from Two or More Record Sources

You form a collection from two or more record sources by including a CROSS clause in a FIND statement RSE:

```
DTR> ! Assume you want to find out which jobs one employee has
DTR> ! held in the company.
DTR> !
DTR> READY EMPLOYEES, JOB_HISTORY
DTR> FIND EMPLOYEES CROSS JOB_HISTORY OVER EMPLOYEE_ID WITH
CON> EMPLOYEE_ID = "00472"
[1 record found]
DTR> PRINT
No record selected, printing whole collection.
```

ID	LAST NAME	FIRST NAME INIT	ADDRESS DATA	ZIP	SEX	SOCIAL SECURITY
00472	Delano	A1	F			
114	Princeton Rd.		Bennington	MA	03442	M 005 89 7164
3/03/29	00472	DSUP	27-Apr-1981		ADMN	00225

```
DTR> ! This is tough to read, a normal occurrence when you are
DTR> ! crossing records. Note that the value 00472 appears twice.
DTR> ! That is because each record in the collection results from
DTR> ! crossing two records, each of which has a value for EMPLOYEE_ID.
DTR> ! You can use the REDUCE statement to get rid of duplicate values
DTR> ! and pare the data down to the fields which interest you.
DTR> !
DTR> REDUCE TO EMPLOYEE_ID, NAME, DEPARTMENT_CODE, JOB_CODE, JOB_START
DTR> PRINT
No record selected, printing whole collection.
```

ID	LAST NAME	FIRST NAME INIT	DEPARTMENT CODE	JOB CODE	JOB START
00472	Delano	A1	F	ADMN	DSUP 27-Apr-1981

```
DTR>
```

The sources for the records you want to cross can be either domains or other collections. Read the section on disadvantages of using collections, however, before you decide to cross collections.

14.7 Removing Records from a Collection

You might create a collection of records with the idea of doing something with the whole group (writing a report, for example) and find that you do not want to use some of the records in the collection. For each record you want to remove from a collection:

- Select the record
- Type PRINT to make sure you selected the right record
- Type DROP to remove that record

The DROP statement does *not* erase the record from storage, only from the collection:

```
DTR> FIND EMPLOYEES WITH LAST_NAME CONTAINING "BURTON"  
[2 records found]  
DTR> PRINT ALL ID, LAST_NAME, FIRST_NAME
```

ID	LAST NAME	FIRST NAME
00237	Burton	Frederick
00417	Burton	Kathleen

```
DTR> SELECT 2  
DTR> PRINT ID, LAST_NAME, FIRST_NAME
```

ID	LAST NAME	FIRST NAME
00417	Burton	Kathleen

```
DTR> DROP  
DTR> SHOW CURRENT  
Collection CURRENT  
Domain: EMPLOYEES  
Number of Records: 2  
Selected Record: 2 (Dropped)
```

```
DTR> PRINT ALL ID, LAST_NAME, FIRST_NAME
```

ID	LAST NAME	FIRST NAME
00237	Burton	Frederick

```
DTR>
```

14.8 Removing Collections from Your Workspace

The `RELEASE` command removes collections from your workspace. `RELEASE ALL` removes all collections. `RELEASE`, followed by one or more collection names, removes the collections you specify:

```
DTR> SHOW COLLECTIONS
Collections:
  COLL      (CURRENT)
  DEG
  EMP
```

```
DTR> RELEASE DEG
DTR> SHOW COLLECTIONS
Collections:
  COLL      (CURRENT)
  EMP
```

```
DTR> RELEASE ALL
DTR> SHOW COLLECTIONS
No established collections.
```

```
DTR>
```

Remember that `RELEASE ALL` removes more than just collections; it also removes from your workspace all declared variables, all loaded tables, and any forms product definitions you have accessed. You load table and form definitions simply by accessing them but variables have to be declared again.

In addition, when you finish a domain from which a collection is formed, you also release the collection. If you need to change the access mode to a domain in order to do something to the records in a collection, ready the domain again with a new access mode. Do not finish it first.

14.9 Disadvantages of Using Collections

There are two disadvantages to using collections:

- There are restrictions that apply to the use of collection-oriented statements in compound statements.

This means that your options are limited when designing procedures that manipulate collections and selected records. You cannot use `FIND`, `SORT`, `REDUCE`, and `DROP` statements in `FOR`, `REPEAT`, `THEN`, `WHILE`, or `BEGIN-END` statements.

As an alternative to the `SORT` and `REDUCE` statements, you can use the `REDUCED TO` and `SORTED BY` clauses in the `RSE` of the `FIND` statement that creates the collection. There is no equivalent for `DROP`, however. In addition, the `SELECT` statement can produce unexpected results when included in a compound statement.

- DATATRIEVE does not use keyed access when a collection is the record source.

This means that all search, cross, and sort operations that manipulate records in collections are done sequentially, even when you base them on fields that are index keys for a data file. If you are processing a collection of 50 or fewer records, the slower performance of sequential searches might not bother you. If you are doing complicated operations on a collection of 500 or more records, the response time might be unacceptable.

To get around this problem, put your key-based operations in the RSE of any FIND statement that creates a large collection from a domain. In addition, avoid creating collections from other collections when the latter contain thousands of records.

For cross operations, use only key-based access. Crossing records in collections can be very time consuming, even when the collections each contain fewer than 50 records.

This discussion should not discourage you from working with collections. As the rest of the chapter indicates, collections provide numerous advantages, especially when you are learning to use DATATRIEVE. You should still read Chapter 15, however, to find out about methods of data retrieval you can use as alternatives to collections. With the combined information in this chapter and in Chapter 15, you can tailor data access to meet both your convenience and performance needs. You can also read the chapter on improving DATATRIEVE performance in the *VAX DATATRIEVE User's Guide* to find out more about using key-based access.



Accessing Data the Expert Way: Without Collections **15**

This chapter explains how you specify records in a compound statement beginning with FOR or in a statement, such as PRINT, that carries out the operation you want to perform.

This chapter also tells you how to define and use **view domains**. A view domain is a data definition that contains a subset of fields from one domain or a combination of fields from two or more domains. Using a view domain, you can get results that otherwise require complex statements.

15.1 Processing Records from Domains Rather Than Collections

There are two reasons you might want to work with records directly from domains:

- You are working with large numbers of records and want fast access.
- You are using compound statements to process records.

15.1.1 Ensuring Fast Access

When you use the FIND statement to create a collection, you are not removing records from files or copying records to some sort of temporary storage area. The FIND statement creates a list of pointer values. For each record in the collection, there is a pointer value that tells where its data is located in a file (or files, if the collection record resulted from a cross operation). Whenever DATATRIEVE must search through records in a collection, it must process every pointer value and look at all the places where associated data is stored.

You can often get DATATRIEVE to respond more quickly when you work directly with domains. You achieve faster response time by specifying a readied domain as the record source and an index key field as the criteria for any searching, sorting, and crossing that you want done. In this case, DATATRIEVE can use the indexes associated with data files to find the records you want. It does not have to check every record in the data file to see which ones meet your needs.

Example 15-1 contrasts three ways to perform the same operation. The operation retrieves records for current employees in a manufacturing department, sorts the records by job code, and displays selected fields from the records. The comment lines in the example begin with an exclamation point (!) and prepare you for the input that follows.

Example 15-1: Including RSEs in Statements

```
DTR> READY JOB_HISTORY
DTR> !
DTR> ! The next input illustrates data retrieval by first
DTR> ! forming a collection and then manipulating and displaying
DTR> ! the data it contains. The first FIND statement uses
DTR> ! key-based access. The second FIND statement and the SORT
DTR> ! statement access records using collection pointer values.
DTR> ! WHO_IS_IT is a domain table that links EMPLOYEE_ID with
DTR> ! EMPLOYEE_NAME.
DTR> !
DTR> FIND JOB_HISTORY WITH DEPARTMENT_CODE = "MBMN"
[36 records found]
DTR> FIND CURRENT WITH JOB_END MISSING
[13 records found]
DTR> SORT BY JOB_CODE
DTR> PRINT ALL JOB_CODE, EMPLOYEE_ID,
CON> EMPLOYEE_ID VIA WHO_IS_IT
```

JOB CODE	EMPLOYEE ID	EMPLOYEE NAME	
APGM	00275	DuBois	Alvin Q
APGM	00449	Leger	Carol
APGM	00167	Kilpatrick	Janet
DGFR	00349	Chandler	Christine E
DMGR	00164	Toliver	Alvin A
DSUP	00344	Kawell	Edward H
EENG	00198	Gehr	Leslie
EENG	00447	Potter	Beverly O
GFER	00329	Rodrigo	Jerry D
MENG	00410	Klein	Walter X
SANL	00366	Harrington	Russ J
SANL	00433	Glackemeyer	Jodie
SPGM	00217	Siciliano	James X

```

DTR> !
DTR> ! The next input performs the same operation using a
DTR> ! compound statement. The FOR statement component restricts
DTR> ! the records to the ones you want and specifies the order in
DTR> ! which you want them. The PRINT statement component specifies
DTR> ! the fields you want to see displayed.
DTR> !
DTR> FOR JOB_HISTORY WITH DEPARTMENT_CODE = "MBMN" AND
CON> JOB_END MISSING SORTED BY JOB_CODE
CON> PRINT JOB_CODE, EMPLOYEE_ID, EMPLOYEE_ID VIA WHO_IS_IT

```

JOB CODE	EMPLOYEE ID	EMPLOYEE NAME
APGM 00275	DuBois	Alvin Q
APGM 00449	Leger	Carol
.	.	.
.	.	.
.	.	.
SPGM 00217	Siciliano	James X

```

DTR> !
DTR> ! The following input includes everything you need to do in
DTR> ! one PRINT statement. Note that when you do this, the record
DTR> ! selection and sorting is specified last, following the
DTR> ! keyword OF. You always specify record selection clauses
DTR> ! last when you put them inside the statement that carries out
DTR> ! the operation you want to perform.
DTR> !
DTR> PRINT JOB_CODE, EMPLOYEE_ID, EMPLOYEE_ID VIA WHO_IS_IT OF
CON> JOB_HISTORY WITH DEPARTMENT_CODE = "MBMN" AND
CON> JOB_END MISSING SORTED BY JOB_CODE

```

JOB CODE	EMPLOYEE ID	EMPLOYEE NAME
APGM 00275	DuBois	Alvin Q
APGM 00449	Leger	Carol
.	.	.
.	.	.
.	.	.
SPGM 00217	Siciliano	James X

```
DTR>
```

15.1.2 Processing Records in Compound Statements

Example 15-2 illustrates another reason for working directly with domains. The example shows two procedures designed to help users modify one or more records in the EMPLOYEES domain. Both procedures use a number of compound statements (statements that contain other statements). In the example, the compound statements start with the keywords FOR, WHILE, IF, and BEGIN. When executed, the first procedure fails because it includes collection-oriented statements inside compound statements. The second procedure works. It substitutes a FOR statement and slightly different statement structure to accomplish the same task.

The comment lines explain the purpose of most input in the two procedures. The section on working with multiple records explains about processing more than one record at a time. Chapter 17 discusses how you create and use procedures and Chapter 18 explains variables.

Example 15-2: Using RSEs in Compound Statements

```
DTR> SHOW MODIFY_AN_EMPLOYEE_RECORD1
PROCEDURE MODIFY_AN_EMPLOYEE_RECORD1
READY EMPLOYEES MODIFY
!
! The DECLARE statements create the variables STILL_WORKING,
! GET_ID, and YES_OR_NO. STILL_WORKING starts out with the
! value "Y". After users modify a record, the procedure prompts
! them to enter a new value for STILL_WORKING. If users enter
! Y in response to this prompt, they can modify another record.
! The reasons for declaring the other variables are explained
! when the procedure uses them.
!
DECLARE STILL_WORKING PIC X.
STILL_WORKING = "Y"
!
DECLARE GET_ID PIC X(5).
!
DECLARE YES_OR_NO PIC X.
!
! The WHILE statement includes all of the statements starting
! with the first BEGIN and concluding with the last END.
! This "outer" BEGIN-END block can execute more than once,
! depending on value entries for STILL_WORKING.
!
WHILE STILL_WORKING CONTAINING "Y"
  BEGIN
  !
  ! The value for the variable GET_ID is assigned by
  ! a prompting value expression. This allows the user to
  ! enter the employee ID for the record needing change.
  !
  GET_ID = *."ID for the employee record you want to change"
  !
  ! The following FIND and SELECT statements should not be
  ! subordinate to the WHILE statement or contained in a
  ! BEGIN-END block. Because they are, the procedure will fail.
  !
  !
```

```

FIND EMPLOYEES WITH EMPLOYEE_ID = GET_ID
SELECT
!
! The following lines assume the presence of a selected
! record. The value for the variable YES_OR_NO is assigned
! by a prompting value expression. The user gets to see the
! record and to decide whether or not it should be modified.
!
PRINT
YES_OR_NO = *."Y if you want to modify this record"
IF YES_OR_NO CONTAINING "Y" THEN
  BEGIN
    PRINT "Press TAB in response to Enter prompts for"
    PRINT "fields you do not want to change.", SKIP
    MODIFY
    PRINT "Changed record now looks like this:", SKIP
    PRINT
  END
  STILL_WORKING = _*."Y if you have more records to change"
END
FINISH EMPLOYEES
PRINT "Access ended to EMPLOYEES."
END_PROCEDURE

```

```

DTR> !
DTR> ! When someone executes the procedure, DATATRIEVE returns
DTR> ! an error message. In this case, no collection was created,
DTR> ! the SELECT statement produced an error, and parts of the
DTR> ! procedure did not execute.
DTR> !
DTR> :MODIFY_AN_EMPLOYEE_RECORD1
No collection for select.
Access ended to EMPLOYEES.

```

```

DTR> !
DTR> ! Here is the procedure revised to work directly with
DTR> ! domains.
DTR> !
DTR> SHOW MODIFY_AN_EMPLOYEE_RECORD2
PROCEDURE MODIFY_AN_EMPLOYEE_RECORD2
READY EMPLOYEES MODIFY
DECLARE STILL_WORKING PIC X.
STILL_WORKING = "Y"
DECLARE GET_ID PIC X(5).
DECLARE YES_OR_NO PIC X.
WHILE STILL_WORKING CONTAINING "Y"
  BEGIN
    GET_ID = *."ID for the employee record you want to change"
    !
    ! Using the domain table WHO_IS_IT, this procedure checks
    ! to ensure that the ID entered by the user exists in the
    ! EMPLOYEES domain. If the ID is not valid, the procedure
    ! prints a message and ignores the FOR statement.
    !

```

(continued on next page)

```

IF GET_ID NOT IN WHO_IS_IT THEN PRINT "No such ID." ELSE
!
! The FOR statement replaces the FIND and SELECT statements to
! specify the record being changed. It tells DATATRIEVE to
! execute all the statements in the following BEGIN-END block
! for each record specified in the FOR statement RSE. In this
! case, that RSE specifies one record.
!
FOR EMPLOYEES WITH EMPLOYEE_ID = GET_ID
BEGIN
LIST
YES_OR_NO = *."Y if you want to modify this record"
IF YES_OR_NO CONTAINING "Y" THEN
BEGIN
PRINT SKIP, "Press TAB in response to Enter prompts for"
PRINT "fields you do not want to change.", SKIP
MODIFY
PRINT SKIP, "Changed record now looks like this:", SKIP
LIST
END
END
STILL_WORKING = *."Y to change more records, N to exit"
END
FINISH EMPLOYEES
PRINT "Access ended to EMPLOYEES records."
END_PROCEDURE

```

```

DTR> !
DTR> ! Here is what happens when the second procedure executes.
DTR> !
DTR> :MODIFY_AN_EMPLOYEE_RECORD2
Enter ID for the employee record you want to change: 0016R
No such ID.
Enter Y to change more records, N to exit: Y
Enter ID for the employee record you want to change: 00164

```

```

EMPLOYEE_ID      : 00164
LAST_NAME        : Toliver
FIRST_NAME       : Alvin
MIDDLE_INITIAL   : A
ADDRESS_DATA     :
STREET           : 146 Parnell Place
TOWN              : Chocorua
STATE            : NH
ZIP              : 03817
SEX              : M
SOCIAL_SECURITY  : 763 08 0064
BIRTHDAY         : 3/28/47
Enter Y if you want to modify this record: Y

```

```

Press TAB in response to Enter prompts for
fields you do not want to change.

```

```
Enter EMPLOYEE_ID: (TAB)
Enter LAST_NAME: (TAB)
Enter FIRST_NAME: (TAB)
Enter MIDDLE_INITIAL: (TAB)
Enter ADDRESS_DATA: (TAB)
Enter STREET: 52 Tiger Way
Enter TOWN: (TAB)
Enter STATE: (TAB)
Enter ZIP: (TAB)
Enter SEX: (TAB)
Enter SOCIAL_SECURITY: (TAB)
Enter BIRTHDAY: (TAB)
```

Changed record now looks like this:

```
EMPLOYEE_ID      : 00164
LAST_NAME        : Toliver
FIRST_NAME       : Alvin
MIDDLE_INITIAL   : A
ADDRESS_DATA     :
STREET           : 52 Tiger Way
TOWN             : Chocorua
STATE            : NH
ZIP              : 03817
SEX              : M
SOCIAL_SECURITY  : 763 08 0064
BIRTHDAY        : 3/28/47
Enter Y to change more records, N to exit: N
Access ended to EMPLOYEES records.
```

DTR>

15.2 Creating RSEs

Chapter 14 listed and described the options you have when including a record selection expression (RSE) in a `FIND` statement. These options and the order in which you can specify them are the same for any statement that can contain an RSE.

Here are some examples of RSEs in FOR, PRINT, and MODIFY statements. So that you can focus on the position of an RSE in the statement, the RSEs are highlighted by gray shading:

```
PRINT FIRST 10 DEGREES
FOR DEGREES SORTED BY DEGREE_FIELD PRINT DEGREE_FIELD, DEGREE,
COLLEGE_CODE
PRINT DEGREES WITH COLLEGE_CODE = "STAN"
FOR SAMPLE IN FIRST 10 DEGREES WITH COLLEGE_CODE = "STAN"
PRINT EMPLOYEE_ID, DEGREE, DEGREE_FIELD
PRINT NAME, ADDRESS OF FIRST 2 EMPLOYEES
MODIFY JOB_END OF JOB_HISTORY WITH JOB_END MISSING AND
EMPLOYEE_ID = "00192"
```

The following example includes all the RSE options in one PRINT statement. The statement joins records stored in three different places to display information about past jobs and salaries for an employee. When you include an RSE in a PRINT statement, as in the example, the order of fields listed in a REDUCED TO clause also specifies the order in which fields are displayed. When you include a REDUCED TO clause in a FOR statement RSE, any subordinate PRINT statement must specify the field display order when it differs from the way those fields are stored in records.

The WITH clause restricts the data to one employee and also matches salary data to a job. Chapter 18 tells you more about using the BETWEEN operator and including more than one condition in a WITH clause.

The SORTED BY clause ensures that the most recent job and salary data displays first:

```
PRINT EMPLOYEES CROSS JOB_HISTORY OVER EMPLOYEE_ID CROSS
SALARY_HISTORY OVER EMPLOYEE_ID WITH
(EMPLOYEE_ID = "00168") AND SALARY_START BETWEEN JOB_START AND
JOB_END) REDUCED TO LAST_NAME, DEPARTMENT_CODE, JOB_CODE,
JOB_START, SALARY_START, SALARY_AMOUNT SORTED BY
DECREASING JOB_START, DECREASING SALARY_START
```

When you get complex RSEs like that one to do your work, you can feel quite accomplished. You are well on your way to being a DATATRIEVE expert. Here is the data that the PRINT statement displays:

LAST NAME	DEPARTMENT CODE	JOB CODE	JOB START	SALARY START	SALARY AMOUNT
Nash	SUWE	PRGM	23-Feb-1979	10-Oct-1981	\$27,126.00
Nash	SUWE	PRGM	23-Feb-1979	15-Oct-1980	\$25,057.00
Nash	SUWE	PRGM	23-Feb-1979	21-Oct-1979	\$23,919.00
Nash	SUWE	PRGM	23-Feb-1979	23-Feb-1979	\$23,605.00
Nash	ENG	PRGM	30-Oct-1977	26-Aug-1978	\$21,520.00
Nash	ENG	PRGM	30-Oct-1977	30-Oct-1977	\$20,883.00
Nash	ELMC	APGM	1-Jul-1975	21-Apr-1977	\$15,977.00
Nash	ELMC	APGM	1-Jul-1975	24-Aug-1976	\$15,851.00
Nash	ELMC	APGM	1-Jul-1975	1-Jul-1975	\$15,179.00

The section later in this chapter on view domains shows you how to use a view domain to display this data in a more readable format.

15.3 Working with Multiple Records

Both Example 15-1 and Example 15-2 specify operations that can be performed on more than one record. When you specify iterative operations, you are creating what is sometimes called a **loop**. In Example 15-1, you specify a group of records and tell DATATRIEVE to do a print operation for each record in the group. The RSE defines a loop because it specifies more than one record for an operation. In Example 15-2, the WHILE statement defines a loop even though the RSE specifies one record. All operations contained in the WHILE statement can execute more than once depending on a variable value under the control of the person executing the procedure.

In these examples, the loops are intentional and help users get work done more quickly and efficiently. There are record processing loops in the following areas, however, that can cause problems for new users:

- FOR statements
- CROSS clauses
- List fields

The following sections discuss these areas in more detail.

15.3.1 FOR Statement Looping Errors

When you include an RSE in a FOR statement, make sure you do not put an RSE in a statement subordinate to the FOR statement. In the following example, the user forgot to enter a SORTED BY clause in the FOR statement RSE and thought the clause could go in a PRINT statement RSE to make up for the oversight. In this case, DATATRIEVE uses the PRINT statement RSE as the record specification and the FOR statement RSE as a counter. It displays the requested information but repeats the display as many times as there are records in COLLEGES:

```
DTR> FOR COLLEGES
CON> PRINT COLLEGE_NAME, TOWN, ZIP OF COLLEGES SORTED BY ZIP
```

```
Bates College          Lewiston          04240
Colby College         Waterville       04563
University of Maine   Orono           04913
.
.
.
.
.
U. of Southern California San Diego       98431
Bates College         Lewiston       04240
Colby College         Waterville     04563
University of Maine   Orono         04913
.
.
.
.
.
U. of Southern California San Diego       98431
Bates College         Lewiston       04240
.
.
.
.
.
```

Either of the following statements would have produced correct results:

```
DTR> FOR COLLEGES SORTED BY ZIP
CON> PRINT COLLEGE_NAME, TOWN, ZIP
```

```
DTR> PRINT COLLEGE_NAME, TOWN, ZIP OF
CON> COLLEGES SORTED BY ZIP
```

15.3.2 CROSS Clause Looping Errors

For each CROSS clause that joins data stored in different locations, include an OVER clause to specify a field DATATRIEVE can use to match records for the join. When you want to limit the values that fields can contain, include a WITH clause:

```
EMPLOYEES CROSS JOB_HISTORY OVER EMPLOYEE_ID CROSS
SALARY_HISTORY OVER EMPLOYEE_ID WITH EMPLOYEE_ID = "00168"
```

If you omit the first OVER clause in the example, you are telling DATATRIEVE to take the first record from EMPLOYEES and join it to each record in JOB_HISTORY, then take the second record from EMPLOYEES and join it to each record in JOB_HISTORY, and so forth. If there are 350 records in EMPLOYEES and 800 records in JOB_HISTORY, DATATRIEVE produces 280,000 hybrid records for the first cross operation. It then takes each of those 280,000 records and uses them for the second cross operation. If you also omit the second OVER clause and SALARY_HISTORY contains 1200 records, DATATRIEVE produces a grand total of 280,000 times 1200 hybrid records.

If you omit any OVER clause, you are asking DATATRIEVE to process many more records than you intended. This will produce results you do not want.

DATATRIEVE interprets the clause OVER EMPLOYEE_ID as WITH EMPLOYEE_ID = EMPLOYEE_ID. In fact, the following RSEs are equivalent to the preceding one. The first RSE names each record source (A, B, and C) and uses these names to qualify the EMPLOYEE_ID fields in each component of the WITH clause. The second RSE uses the top-level field names in each record source to qualify the EMPLOYEE_ID fields:

```
A IN EMPLOYEES CROSS B IN JOB_HISTORY
CROSS C IN SALARY_HISTORY WITH (B.EMPLOYEE_ID = A.EMPLOYEE_ID) AND
(C.EMPLOYEE_ID = B.EMPLOYEE_ID)
```

```
EMPLOYEES CROSS JOB_HISTORY CROSS SALARY_HISTORY WITH
(JOB_HISTORY_REC.EMPLOYEE_ID = EMPLOYEES_REC.EMPLOYEE_ID) AND
(SALARY_HISTORY_REC.EMPLOYEEID = JOB_HISTORY_REC.EMPLOYEE_ID)
```

Before you enter an RSE that includes one or more CROSS clauses, check your input to make sure you included a corresponding number of OVER clauses or a corresponding number of equivalent conditions in a WITH clause. If you inadvertently start a runaway cross operation, you can enter CTRL/C to stop it.

15.3.3 Lists, the “Record” Within the Record

The preceding two sections discussed looping problems users learn to avoid. This section discusses looping users must learn to include.

When a record definition includes a list field (defined by the OCCURS clause), it means that fields subordinate to the list field can contain more than one value (or occurrence) per record. If you need to access a particular value in a list field, perhaps because you want to change it, you must create a loop to get at it. You do this by treating the list field as you would a record source, so that DATATRIEVE can recognize and process the fields it contains. The domains from the sample personnel system in this book do not contain records with list fields. The FAMILIES domain in CDD\$TOP.DTR\$LIB.DEMO does. The following example uses the FAMILIES domain to illustrate how you can access values in list fields:

```
DTR> READY FAMILIES MODIFY
DTR> PRINT FIRST 5 FAMILIES
```

FATHER	MOTHER	NUMBER KIDS	KID 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	1
ARNIE	ANNE	2	SCOTT	20
			BRIAN	20

```
DTR> !
DTR> ! The SHOW FIELDS command reveals that the group field
DTR> ! EACH_KID and the elementary fields KID_NAME and AGE
DTR> ! are subordinate to the list field KIDS.
```

```
DTR> !
DTR> SHOW FIELDS FOR FAMILIES
```

```
FAMILIES
  FAMILY
    PARENTS
      FATHER    <Character string>
      MOTHER    <Character string>
    NUMBER_KIDS <Number>
    KIDS        <List>
      EACH_KID
        KID_NAME (KID) <Character string>
        AGE      <Number>
```

```
DTR> !
DTR> ! If you try to use a list field as a field name, it
DTR> ! does not work. Neither can you refer to list field
DTR> ! subordinates as you do to other fields in the record.
```

```
DTR> !
DTR> PRINT KIDS OF FIRST 1 FAMILIES
PRINT KIDS OF FIRST 1 FAMILIES
^
```

Expected end of statement, encountered "OF".

DTR> PRINT EACH_KID OF FIRST 1 FAMILIES

"EACH_KID" is undefined or used out of context.

DTR> !

DTR> ! If you want to put an RSE in a PRINT statement, you can

DTR> ! set up a double loop by putting two RSEs at the end of

DTR> ! the statement--the first for the list field and the second

DTR> ! for the domain. For each OF RSE clause, put an ALL before

DTR> ! the field name (or list of field names) that you want to

DTR> ! display.

DTR> !

DTR> PRINT ALL ALL EACH_KID OF KIDS OF FIRST 1 FAMILIES

KID NAME	AGE
-------------	-----

URSULA	7
--------	---

RALPH	3
-------	---

DTR> !

DTR> ! You can also set up a list field loop when you use the

DTR> ! FOR statement. In this case, after the first FOR statement

DTR> ! that contains the RSE for the domain, enter a second FOR

DTR> ! statement that contains the RSE for the list. (If you were

DTR> ! trying to get at an item in a list field subordinate to

DTR> ! another list field, you would need three FOR statements--

DTR> ! one for the domain, one for the outer list, and one for the

DTR> ! inner list.)

DTR> !

DTR> FOR FIRST 1 FAMILIES

CON> FOR KIDS

CON> PRINT EACH_KID

KID NAME	AGE
-------------	-----

URSULA	7
--------	---

RALPH	3
-------	---

DTR> !

DTR> ! The following input shows FOR statement looping that lets

DTR> ! you change values occurring in list fields.

DTR> !

DTR> FOR FIRST 1 FAMILIES

CON> FOR KIDS

CON> MODIFY AGE

Enter AGE: 8

Enter AGE: 4

DTR> PRINT FIRST 1 FAMILIES

FATHER	MOTHER	NUMBER KIDS	KID NAME	AGE
JIM	ANN	2	URSULA	8
			RALPH	4

DTR>

15.4 Creating Views

A view is a data definition that specifies fields from one or more domains. You ready a view just as you do a domain and can display or modify data in the fields it contains. The only operation you cannot do using a view is storing records in the associated domains. To store records in a domain, you must ready the domain itself.

You can define a view so that users can access a subset of fields from a long record. You might want to do this for one of two reasons:

- The records you want to access contain more fields than you want to use. Without a view, you must include a list of fields in PRINT and MODIFY statements to restrict data display and access. After you define the view, you can simply use the view name in READY and PRINT statements to get the access you need.
- You want users to be able to access a file that contains some data they have no right to see. In this case, you can define a view that specifies the fields these users are allowed to see.

You can also define a view so that you or other users can access data stored in more than one place. In this case, the view performs what would otherwise require one or more CROSS clauses in a fairly complex statement.

15.4.1 View Domains That Subset Fields from One Domain

Example 15-3 shows you how to define and use a view containing a subset of fields from one domain. As you will see, a view definition has a combination of features you have used before when creating domain and record definitions.

Example 15-3: Defining and Using a View

```
DTR> ! Your view definition must start with the keywords
DTR> ! DEFINE DOMAIN followed by the name you choose for the view.
DTR> ! The OF clause specifies the name of the domain on which
DTR> ! the view is based. The keyword USING is optional.
DTR> !
DTR> DEFINE DOMAIN COLLEGE_LOCATIONS OF COLLEGES USING
DFN> !
DFN> ! Your first level number specifies a field that
DFN> ! DATATRIEVE uses to identify an RSE from the domain. You can
DFN> ! pick any name you want for this field. Follow the name
DFN> ! with the keywords OCCURS FOR and then enter the RSE. In
DFN> ! this case the RSE includes only the domain name. End this
DFN> ! and all subsequent field definitions with a period.
DFN> !
```

```

DFN> 01 COLLEGE OCCURS FOR COLLEGES.
DFN> !
DFN> ! Using level numbers higher than the field that identifies
DFN> ! the domain RSE, put in field definitions for each of the
DFN> ! fields from records in the RSE that you want in the view.
DFN> ! Make sure you use the same name that identifies the field
DFN> ! in the source domain. Each field definition must include a
DFN> ! FROM clause that specifies the source domain.
DFN> !
DFN> 03 COLLEGE_NAME FROM COLLEGES.
DFN> 03 TOWN FROM COLLEGES.
DFN> 03 STATE FROM COLLEGES.
DFN> 03 ZIP FROM COLLEGES.
DFN> !
DFN> ! End the view definition with a semicolon to let DATATRIEVE
DFN> ! know you have finished it.
DFN> !
DFN> ;
DTR> SHOW COLLEGE_LOCATIONS
DOMAIN COLLEGE_LOCATIONS OF COLLEGES USING
01 COLLEGE OCCURS FOR COLLEGES.
03 COLLEGE_NAME FROM COLLEGES.
03 TOWN FROM COLLEGES.
03 STATE FROM COLLEGES.
03 ZIP FROM COLLEGES.

```

```

DTR> READY COLLEGE_LOCATIONS
DTR> SHOW READY
Ready sources:
COLLEGE_LOCATIONS: Domain, VIEW, protected read
<CDD$TOP.PERSONNEL.COLLEGE_LOCATIONS;1>
No loaded tables.

```

```
DTR> PRINT COLLEGE_LOCATIONS
```

COLLEGE NAME	TOWN	STATE	ZIP
American University	Washington, DC		20073
Bates College	Lewiston	ME	04240
Bowdoin College	Brunswick	ME	04913
Cal. Institute of Tech.	Pasadena	CA	91342
Colby College	Waterville	ME	04563
.	.	.	.
.	.	.	.
.	.	.	.
Yale University	New Haven	CT	06510

```
DTR>
```

Here is an example of a more complex view definition. Even though the view specifies fields from only one domain, DATATRIEVE must use two domains to resolve the RSE identified by CURRENT_FOLKS. Therefore, both domains must be listed in the OF clause:

```
DTR> SHOW ADDRESS_LIST
DOMAIN ADDRESS_LIST OF EMPLOYEES, JOB_HISTORY USING
01 CURRENT_FOLKS OCCURS FOR EMPLOYEES CROSS
   JOB_HISTORY OVER EMPLOYEE_ID WITH JOB_END MISSING.
03 LAST_NAME      FROM EMPLOYEES.
03 FIRST_NAME     FROM EMPLOYEES.
03 MIDDLE_INITIAL FROM EMPLOYEES.
03 ADDRESS_DATA   FROM EMPLOYEES.
03 STREET         FROM EMPLOYEES.
03 TOWN          FROM EMPLOYEES.
03 ZIP           FROM EMPLOYEES.
;
```

```
DTR> READY ADDRESS_LIST
DTR> LIST ADDRESS_LIST
```

```
LAST_NAME      : Toliver
FIRST_NAME     : Alvin
MIDDLE_INITIAL : A
ADDRESS_DATA   :
STREET        : 146 Parnell Place
TOWN          : Chocorua
ZIP           : 03817
```

```
LAST_NAME      : Smith
FIRST_NAME     : Terry
MIDDLE_INITIAL : D
ADDRESS_DATA   :
STREET        : 120 Tenby Dr.
TOWN          : Chocorua
ZIP           : 03817
```

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

```
DTR>
```

15.4.2 View Domains That Combine Fields from Two or More Domains

The following example illustrates a view definition that specifies fields from more than one domain. It allows users to access both current and historical job and salary information for current employees:

```
DTR> SHOW EMPLOYEE_HISTORY_1
DOMAIN EMPLOYEE_HISTORY_1 OF EMPLOYEES, SALARY_HISTORY,
JOB_HISTORY USING
Ø1 ONE_OF_US OCCURS FOR EMPLOYEES CROSS JOB_HISTORY OVER
EMPLOYEE_ID WITH JOB_END MISSING.
Ø3 EMPLOYEE_ID FROM EMPLOYEES.
Ø3 LAST_NAME FROM EMPLOYEES.
Ø3 JOBS_HERE OCCURS FOR JOB_HISTORY WITH EMPLOYEE_ID =
EMPLOYEES_REC.EMPLOYEE_ID SORTED BY DECREASING JOB_START.
Ø5 JOB_CODE FROM JOB_HISTORY.
Ø5 DEPARTMENT_CODE FROM JOB_HISTORY.
Ø5 JOB_START FROM JOB_HISTORY.
Ø3 SALARIES OCCURS FOR SALARY_HISTORY WITH
(EMPLOYEE_ID = EMPLOYEES_REC.EMPLOYEE_ID) SORTED BY
DECREASING SALARY_START.
Ø5 SALARY_START FROM SALARY_HISTORY.
Ø5 SALARY_AMOUNT FROM SALARY_HISTORY.
;

DTR> READY EMPLOYEE_HISTORY_1
DTR> !
DTR> ! Note that DATATRIEVE considers fields subordinate to all but
DTR> ! the first OCCURS field as list items.
DTR> !
DTR> SHOW FIELDS FOR EMPLOYEE_HISTORY_1
EMPLOYEE_HISTORY_1
ONE_OF_US
EMPLOYEE_ID (ID) <Character string, indexed key>
LAST_NAME (L_NAME) <Character string, indexed key>
JOBS_HERE <List>
JOB_CODE (JOB) <Character string>
DEPARTMENT_CODE (DEPT) <Character string>
JOB_START <Date>
SALARIES <List>
SALARY_START <Date>
SALARY_AMOUNT (SALARY) <Number, indexed key>

DTR> PRINT EMPLOYEE_HISTORY_1 WITH EMPLOYEE_ID = "ØØ168"
```

(continued on next page)

ID	LAST NAME	JOB CODE	DEPARTMENT CODE	JOB START	SALARY START	SALARY AMOUNT
00168	Nash	SPGM	MGVT	18-Feb-1982	15-Dec-1982	\$32,254.00
		PRGM	SUWE	23-Feb-1979	18-Feb-1982	\$29,469.00
		PRGM	ENG	30-Oct-1977	10-Oct-1981	\$27,126.00
		APGM	ELMC	1-Jul-1975	15-Oct-1980	\$25,057.00
					21-Oct-1979	\$23,919.00
					23-Feb-1979	\$23,605.00
					26-Aug-1978	\$21,520.00
					30-Oct-1977	\$20,883.00
					21-Apr-1977	\$15,977.00
					24-Aug-1976	\$15,851.00
			1-Jul-1975	\$15,179.00		

DTR>

The following view is very similar to the first one but it associates salary data with job data. Note that the level numbers indicate the subordinate relationship of salary fields to job fields. In addition, the WITH clause in the last OCCURS RSE restricts salary values to the ones current for a given job.

Because the missing value specified for JOB_END is earlier than any other dates (November 17, 1858), the BT (BETWEEN) operator cannot detect salaries for the current job in the sample database. This view, therefore, omits current job and salary data from the display (JOBS_HERE... WITH JOB_END NOT MISSING):

```
DTR> SHOW EMPLOYEE_HISTORY_2
DOMAIN EMPLOYEE_HISTORY_2 OF EMPLOYEES, SALARY_HISTORY,
      JOB_HISTORY USING
01 ONE_OF_US OCCURS FOR EMPLOYEES CROSS JOB_HISTORY OVER
      EMPLOYEE_ID WITH JOB_END MISSING.
03 EMPLOYEE_ID FROM EMPLOYEES.
03 LAST_NAME FROM EMPLOYEES.
03 JOBS_HERE OCCURS FOR JOB_HISTORY WITH
      (EMPLOYEE_ID = EMPLOYEES_REC.EMPLOYEE_ID) AND
      (JOB_END NOT MISSING) SORTED BY DECREASING JOB_START.
05 JOB_CODE FROM JOB_HISTORY.
05 DEPARTMENT_CODE FROM JOB_HISTORY.
05 JOB_START FROM JOB_HISTORY.
05 SALARIES_FOR_JOBS OCCURS FOR SALARY_HISTORY WITH
      (EMPLOYEE_ID = JOB_HISTORY_REC.EMPLOYEE_ID) AND
      (SALARY_START BT JOB_START AND JOB_END) SORTED BY
      DECREASING SALARY_START.
07 SALARY_START FROM SALARY_HISTORY.
07 SALARY_AMOUNT FROM SALARY_HISTORY.
```

```
DTR> READY EMPLOYEE_HISTORY_2
DTR> PRINT EMPLOYEE_HISTORY_2 WITH EMPLOYEE_ID = "00168"
```

ID	LAST NAME	JOB CODE	DEPARTMENT CODE	JOB START	SALARY START	SALARY AMOUNT
00168	Nash	PRGM	SUWE	23-Feb-1979	10-Oct-1981	\$27,126.00
					15-Oct-1980	\$25,057.00
					21-Oct-1979	\$23,919.00
		PRGM	ENG	30-Oct-1977	23-Feb-1979	\$23,605.00
					26-Aug-1978	\$21,520.00
					30-Oct-1977	\$20,883.00
		APGM	ELMC	1-Jul-1975	21-Apr-1977	\$15,977.00
					24-Aug-1976	\$15,851.00
					1-Jul-1975	\$15,179.00

DTR>

If you look back at the earlier section in this chapter on creating RSEs, you will see that the data this view displays is very similar to the data you get from crossing the same three domains in one RSE. It is important to remember that DATATRIEVE considers fields defined in the second and subsequent OCCURS clauses in a view domain as repeating fields. Because of this, you must use the techniques for retrieving list items when you retrieve fields subordinate to the second and subsequent OCCURS fields.

The section on print statement options in Chapter 19 shows an example of a PRINT statement that retrieves such fields from a view domain.

15.5 Access Privileges Needed for Using Views

To ready a view for any task, users need the appropriate ACL privileges from the following list:

- The view itself
- The directory in which the view is located
- Each domain that the view accesses

View users also need the appropriate VMS privileges for all data files associated with the domains on which the view is based and for the VMS directories storing those files.

In short, users cannot ready the view if they do not have sufficient privileges to ready the domains on which the view is based.

Chapter 7 discusses access privileges in greater detail.

15.6 Summary of Options: Advantages and Disadvantages

As you have learned, DATATRIEVE gives you many options for accessing data. This section summarizes those options and their advantages and disadvantages.

- Using the statements that create and manipulate collections

These statements begin with the keywords `FIND`, `SELECT`, `SORT`, `REDUCE`, and `DROP`. The advantage you have when you create a collection is that the RSE in the `FIND` statement is still at your disposal after the `FIND` statement executes. Therefore, the RSE does not have to be exact and you can refine it with other statements after you take a look at the records it specifies.

You do not have the advantage of indexed access to records in a collection. If the collection contains many records and you need to perform complex operations on those records, DATATRIEVE performance is going to be slower than if you used key-based access to a domain. On the other hand, if the collection gathers together only a few records from a domain that contains thousands of records, you might get faster performance using one or more collections as the basis for your operations.

The performance factor aside, you either cannot or should not use collection-oriented statements in compound statements. In most procedures, this limits what you can do with collections.

- Using RSEs in statements other than `FIND`:

This option gives you the greatest flexibility. You can specify the records you want to process in compound statements. You can specify either collections or domains as the record sources in the RSE. To get the best response time from DATATRIEVE, include in the RSE key-based access to a domain or use a small collection when its records come from very large domains.

The disadvantage of using RSEs in statements other than `FIND` is that you must learn to tell DATATRIEVE what records you want in one RSE. Remember, however, you can type `EDIT` immediately after an incorrect statement to correct your mistakes. This takes much of the pain out of learning to enter complex RSEs.

- Defining and using views

A view is an excellent substitute for RSEs and lists of fields that you frequently enter. A view also provides a way to mask sensitive data in a domain from users who should not see it.

One disadvantage of views is that you cannot use them to store records in domains, only to display or modify records already stored.

When a view contains more than one OCCURS RSE, the fields subordinate to all but the first OCCURS field are treated as list items. Accessing these fields individually requires the same method you must use to access list fields defined in record definitions. (See the section earlier in this chapter on lists.) Views that contain more than one OCCURS RSE, in order to be easy to use, should include data that you want to display or modify as a unit.



Maintaining Data 16

This chapter discusses storing, erasing, and modifying records stored in VAX RMS (Record Management Services) data files.

16.1 Storing Records

To store records in a domain, you must first ready it for either write or extend access. If you choose write access, you can also print and modify records in the domain. Ready the domain with the shared option if you know other users might be storing, modifying, or erasing records in that domain at the same time you are. For example:

```
DTR> READY EMPLOYEES SHARED WRITE
DTR>
```

Example 16-1 illustrates two variations of a STORE operation.

Example 16-1: Storing Records Interactively

```
DTR> READY EMPLOYEES SHARED WRITE
DTR> !
DTR> ! To store one record, enter STORE followed by the domain name.
DTR> ! DATATRIEVE then prompts you to enter a value for each of the
DTR> ! elementary fields in the record. After you enter a value,
DTR> ! DATATRIEVE checks the record definition to make sure the
DTR> ! value meets all the field requirements. If the value does not
DTR> ! meet those requirements, DATATRIEVE displays a message and
DTR> ! prompts you to enter another value for the field.
DTR> !
```

(continued on next page)

```

DTR> STORE EMPLOYEES
Enter EMPLOYEE_ID: 00502
Enter LAST_NAME: SCHULTZ
Enter FIRST_NAME: BONNIE
Enter MIDDLE_INITIAL: T
Enter ADDRESS_DATA: RFD 5
Enter STREET: STORK DRIVE
Enter TOWN: HUDSON
Enter STATE: MASS
Truncation during assignment.
Re-enter STATE: MA
Enter ZIP: (TAB)
Enter SEX: F
Enter SOCIAL_SECURITY: 234796666
Enter BIRTHDAY: (TAB)
DTR> !
DTR> ! If you want to store more than one record, you can enter
DTR> ! REPEAT, followed by the number of records you want to store
DTR> ! and a STORE statement. DATATRIEVE then prompts you to enter
DTR> ! field values for the specified number of records.
DTR> !
DTR> REPEAT 3 STORE EMPLOYEES
Enter EMPLOYEE_ID: 00503
Enter LAST_NAME: BRAVO
Enter FIRST_NAME: MARYANN
Enter MIDDLE_INITIAL: M
.
.
.
DTR> !
DTR> ! If you discover a mistake in a previous field entry for the
DTR> ! record you are storing, you can enter CTRL/Z. Then
DTR> ! DATATRIEVE will not store that record. CTRL/Z also
DTR> ! aborts (terminates) a compound statement. If the store
DTR> ! operation is part of a REPEAT statement and you enter
DTR> ! CTRL/Z, you do not get to store any subsequent records
DTR> ! you otherwise would be prompted for.
DTR> !
Enter LAST_NAME: PHARES
Enter FIRST_NAME: ELIZABETH
Enter MIDDLE_INITIAL: C
Enter ADDRESS_DATA: 37 TINTON AVE.
Enter STREET: (CTRL/Z)
^Z
Execution terminated by operator.
DTR>

```

Example 16-2 shows a store operation in a procedure. The example also illustrates the optional USING and VERIFY USING clauses of the STORE statement.

Example 16-2: Storing Records in a Procedure

```
DTR> !  
DTR> ! NEW_PEOPLE is a procedure to be used by people who do not  
DTR> ! understand the restrictions they must obey when entering  
DTR> ! data and who do not know much about DATATRIEVE.  
DTR> !  
DTR> SHOW NEW_PEOPLE  
PROCEDURE NEW_PEOPLE  
!  
! SET ABORT makes sure the procedure will stop if for some  
! reason a user cannot ready the domain.  
!  
SET ABORT  
READY EMPLOYEES SHARED WRITE  
!  
! SET NO ABORT makes sure the procedure does not stop if  
! someone enters CTRL/Z to negate a mistake made while  
! storing a record.  
!  
SET NO ABORT  
!  
! The value for the variable MORE_RECORDS determines when  
! the procedure stops. MORE_RECORDS starts with the value "Y"  
! (for "Yes") and the user gets to change it after storing  
! each record.  
!  
DECLARE MORE_RECORDS PIC X.  
MORE_RECORDS = "Y"  
WHILE MORE_RECORDS CONT "Y"  
BEGIN  
    STORE EMPLOYEES USING  
    BEGIN  
        !  
        ! EMPLOYEE_ID is the primary key for a data file. This  
        ! statement automatically gives it a number that is one  
        ! more than the largest existing value. Because the field  
        ! is defined as PIC X(5), you want to be sure that values  
        ! containing fewer than 5 digits are stored with leading  
        ! zeros. The edit string in the FORMAT value expression  
        ! ensures that this is done.  
        !  
        EMPLOYEE_ID = FORMAT(MAX ID OF EMPLOYEES + 1) USING 99999  
        !  
        ! The following statements display information to help the  
        ! user enter data correctly. The expressions containing an  
        ! asterisk (*) prompt the user to enter a value for a  
        ! field.  
        !  
    END  
END
```

(continued on next page)


```

PRINT "Press the TAB key if you have no data for a field.",
SKIP
LAST_NAME = *."last name"
FIRST_NAME = *."first name"
MIDDLE_INITIAL = *."middle initial"
ADDRESS_DATA = *."RFD number or other misc. address data"
STREET = *."street"
TOWN = *."city or town"
STATE = *."two-character state abbreviation"
SEX = *."sex (M or F)"
BIRTHDAY = *."birth date (11 character maximum)"
!
! You need to let users know when they cannot enter TAB.
! SOCIAL_SECURITY is an index key field that does not allow
! duplicates. You do not want users to leave such a field
! blank. If a user leaves the field blank for one record,
! then subsequent blank entries are considered duplicates.
!
PRINT SKIP,
    "You must enter the employee's social security number.",
SKIP
SOCIAL_SECURITY = *."social security number (digits only)"
END VERIFY USING
!
! The VERIFY clause of the STORE statement lets you put in
! additional statements you want DATATRIEVE to execute
! before it stores a record. In this case, it contains
! an IF statement to make sure the user has not entered
! any spaces or pressed the TAB key for the SOCIAL_SECURITY
! field.
!
IF SOCIAL_SECURITY CONTAINING "" THEN
BEGIN
    PRINT SKIP, "Do not press TAB or enter spaces for this field"
    SOCIAL_SECURITY = _*."social security number again"
END
PRINT SKIP
MORE_RECORDS = *."Y if storing more records, N if not"
END
FINISH EMPLOYEES
PRINT "NEW_PEOPLE program finished."
END_PROCEDURE
DTR) !
DTR) ! Here is what happens when someone executes NEW_PEOPLE.

DTR) !
DTR) :NEW_PEOPLE
Press the TAB key if you have no data for a field.

Enter last name: Warren
Enter first name: Leslie
Enter middle initial: A
Enter RFD number or other misc. address data: (TAB)
Enter street: 16 Ricky Lane
Enter city or town: Hudson
Enter two-character state abbreviation: MA
Enter sex (M or F): F
Enter birth date (11 character maximum): 3/28/46

```

You must enter the employee's social security number.

Enter social security number (digits only): **TAB**

Do not press TAB or enter spaces for this field.

Enter social security number again: **149873444**

Enter Y if storing more records, N if not: **N**

NEW_PEOPLE program finished.

DTR>

You cannot store records in a domain by accessing a view based on that domain. Access domains directly when you want to store records.

16.2 Erasing Records

You must first ready a domain with write access before you can erase any records it contains. Use the shared option if you want to let other users store, erase, or modify records in the domain at the same time you are accessing it.

There are three ways you can erase records. You can erase:

- A selected record in a collection
- All of the records in the CURRENT collection
- All of the records in an RSE

You cannot erase records in a view based on more than one domain or records specified by an RSE that contains a CROSS clause. Although you can erase records in a view that contains a subset of fields from more than one domain, remember that you are erasing all the fields in those records, not just the ones you see in the view. The same holds true for a collection record that results from a REDUCED TO clause or a REDUCE statement.

If you want to delete only one or a few records, it is easiest to isolate records in a collection. Example 16-3 illustrates how to erase records using a collection.

Example 16-3: Erasing Records by First Creating a Collection

```
DTR> ! Erase one of the DEGREES records for the employee with
DTR> ! ID number 00183.
DTR> !
DTR> READY DEGREES SHARED WRITE
DTR> FIND DEGREES WITH EMPLOYEE_ID = "00183"
[5 records found]
DTR> PRINT
No record selected, printing whole collection.
```

(continued on next page)

EMPLOYEE ID	COLLEGE CODE	DEGREE	DEGREE FIELD	DATE GIVEN
00183		Associates	Arts	3-Jul-1964
00183		Masters	Elect. Engrg.	16-Aug-1965
00183		Masters	Applied Math	3-Jul-1965
00183		Bachelors	Arts	14-Jun-1965
00183	MIT	Ph.D.	Elect. Engrg.	20-May-1965

DTR> !
DTR> ! The first record in the collection is the one to be erased.
DTR> !
DTR> SELECT 1
DTR> PRINT

EMPLOYEE ID	COLLEGE CODE	DEGREE	DEGREE FIELD	DATE GIVEN
00183		Associates	Arts	3-Jul-1964

DTR> ERASE
DTR> !
DTR> ! The SHOW CURRENT command indicates the selected record
DTR> ! has been erased. Remember, however, the address pointer
DTR> ! value for that record is still part of the collection.
DTR> ! Therefore, the value for "number of records" in the
DTR> ! collection always stays the same, no matter how many
DTR> ! records you erase from the data file.
DTR> !
DTR> SHOW CURRENT
Collection CURRENT
Domain: DEGREES
Number of Records: 5
Selected Record: 1 (Erased)

DTR> PRINT ALL

EMPLOYEE ID	COLLEGE CODE	DEGREE	DEGREE FIELD	DATE GIVEN
00183		Masters	Elect. Engrg.	16-Aug-1965
00183		Masters	Applied Math	3-Jul-1965
00183		Bachelors	Arts	14-Jun-1965
00183	MIT	Ph.D.	Elect. Engrg.	20-May-1965

DTR> !
DTR> ! The first record in the PRINT ALL display is still
DTR> ! ordinal position 2 in the collection. Ordinal position 1
DTR> ! still belongs to the erased record. This is important to
DTR> ! remember if you are working with collections from which
DTR> ! you have erased records. If your SELECT statement
DTR> ! specifies an ordinal position that belongs to an erased
DTR> ! record, DATATRIEVE tells you it cannot find the record.
DTR> !
DTR> SELECT 1
Selected record not found.

```

DTR> !
DTR> ! If you want to erase all the records in the CURRENT
DTR> ! collection, simply enter ERASE ALL.
DTR> !
DTR> FIND DEGREES WITH EMPLOYEE_ID = "00489"
[3 records found]
DTR> PRINT
No record selected, printing whole collection.

```

EMPLOYEE ID	COLLEGE CODE	DEGREE	DEGREE FIELD	DATE GIVEN
00489		Bachelors	Arts	11-Jun-1983
00489		Masters	Elect. Engrg.	9-Mar-1983
00489	MIT	Masters	Applied Math	11-Jun-1983

```

DTR> ERASE ALL
DTR> PRINT ALL
DTR>

```

Example 16-4 is a procedure that erases records specified in a FOR statement RSE. It is designed to handle a situation that sometimes arises in a personnel system – “new hires” who, because they worked for a company some time ago and then worked at another firm for a while, end up with two employee identification numbers when they return to the original company. Employee identification numbers are usually primary keys for data files and cannot be modified. Therefore, all employee records to which you want to assign new numbers must be erased and stored again.

Note

The sample personnel system in this book defines the SOCIAL_SECURITY field in the EMPLOYEES domain as a key field in which duplicates are not allowed. If users tried to enter a second EMPLOYEE_ID for an employee, the duplicate SOCIAL_SECURITY value would result in an error message and the record would not be stored. The record definitions for all other domains containing the EMPLOYEE_ID field check to make sure the EMPLOYEE_ID exists in the EMPLOYEES domain before records are stored (VALID IF EMPLOYEE_ID IN WHO_IS_IT).

When looking at this example, assume that the SOCIAL_SECURITY field is not a key field and that the appropriate VALID IF clauses are not in place in the record definitions. This is a good illustration of how mistakes can mushroom when database design is faulty.

Example 16-4: Erasing Records Using a FOR Statement RSE

```
DTR> SHOW WASTE_BASKET
PROCEDURE WASTE_BASKET
SET ABORT
READY EMPLOYEES SHARED WRITE, SALARY_HISTORY SHARED WRITE,
JOB_HISTORY SHARED WRITE, DEGREES SHARED WRITE
!
PRINT SKIP, "This program checks the main employee file and also"
PRINT "job, salary, and educational history files to find"
PRINT "records for an employee who has two ID numbers.", SKIP
!
! The display produced by the next block of statements prints the
! two records. Because EMPLOYEE_ID is the primary key for the file
! and the primary key values determine the default order in which
! records are displayed, the lowest (and first EMPLOYEE_ID value
! assigned to the employee) is displayed first.
!
DECLARE GET_SS PIC 9(9).
GET_SS = *."social security number (digits only)"
FIND EMPLOYEES WITH SOCIAL_SECURITY = GET_SS
PRINT EMPLOYEE_ID, SOCIAL_SECURITY, LAST_NAME, STREET, TOWN,
STATE OF CURRENT
!
PRINT SKIP, "Enter the FIRST EMPLOYEE ID in the display in response"
PRINT "to the following prompt. If only one record is displayed or"
PRINT "if the employee records are not the ones you want, terminate"
PRINT "the procedure by entering CTRL/Z."
DECLARE GET_GOOD_ID PIC 9(5).
GET_GOOD_ID = *."employee number for records to be retained"
!
PRINT SKIP, "Enter the LAST EMPLOYEE ID in the display in response"
PRINT "to the following prompt."
DECLARE GET_BAD_ID PIC 9(5).
GET_BAD_ID = *."employee number for records to be erased"
!
! Each of the following FOR statements prints any bad records to
! a file in the user's default VMS directory before erasing
! them. A file is not created and no records are erased if the
! domain in the FOR statement RSE has no records containing
! the value in GET_BAD_ID.
!
FOR EMPLOYEES WITH EMPLOYEE_ID = GET_BAD_ID
BEGIN
  ON STOREEMP.LIS
  BEGIN
    PRINT "Check the employee record with the following ID to"
    PRINT "make sure name and address is current:"
    PRINT SKIP, GET_GOOD_ID
    PRINT SKIP, "Here is the current information:", SKIP
    PRINT LAST_NAME, FIRST_NAME, MIDDLE_INITIAL
    PRINT ADDRESS
  END
  ERASE
END
```

```

!
FOR JOB_HISTORY WITH EMPLOYEE_ID = GET_BAD_ID
BEGIN
  ON STOREJOB.LIS
  BEGIN
    PRINT "These records must be stored in JOB_HISTORY under
    PRINT "the following ID, not the one listed:"
    PRINT SKIP, GET_GOOD_ID, SKIP
    LIST
  END
  ERASE
END
!
FOR SALARY_HISTORY WITH EMPLOYEE_ID = GET_BAD_ID
BEGIN
  ON STORESAL.LIS
  BEGIN
    PRINT "Store these records in SALARY_HISTORY under the
    PRINT "following ID, not the one listed:"
    PRINT SKIP, GET_GOOD_ID, SKIP
    LIST
  END
  ERASE
END
!
FOR DEGREES WITH EMPLOYEE_ID = GET_BAD_ID
BEGIN
  ON STOREDEG.LIS
  BEGIN
    PRINT "Add these records to DEGREES if they are not"
    PRINT "already stored under the following ID:"
    PRINT SKIP, GET_GOOD_ID, SKIP
    PRINT COLLEGE_CODE, DEGREE, DEGREE_FIELD, DATE_GIVEN
  END
  ERASE
END
!
PRINT "If any of the following files are in your default"
PRINT "VMS directory, they contain additional changes"
PRINT "you might have to make to the Personnel database"
PRINT "for this employee:"
PRINT SKIP, "STOREEMP.LIS"
PRINT "STOREJOB.LIS"
PRINT "STORESAL.LIS"
PRINT "STOREDEG.LIS"
FINISH EMPLOYEES, JOB_HISTORY, SALARY_HISTORY, DEGREES
END_PROCEDURE

DTR> !
DTR> ! Here is what happens when a user executes WASTE_BASKET.
DTR> !
DTR> :WASTE_BASKET

```

(continued on next page)

This program checks the main employee file and also job, salary, and educational history files to find records for an employee who has two ID numbers.

Enter social security number (digits only): 269212608

ID	SOCIAL SECURITY	LAST NAME	STREET	TOWN
00173	269 21 2608	Bartlett	149 Steeple Lane	Troy
NH				
00502	269 21 2608	Bartlett	149 Steeple Lane	Troy
NH				

Enter the FIRST EMPLOYEE ID in the display in response to the following prompt. If only one record is displayed or if the employee records are not the ones you want, terminate the procedure by entering CTRL/Z.

Enter employee number for records to be retained: 00173

Enter the LAST EMPLOYEE ID in the display in response to the following prompt.

Enter employee number for records to be erased: 00502

If any of the following files are in your default VMS directory, they contain additional changes you might have to make to the Personnel database for this employee:

STOREEMP.LIS
STOREJOB.LIS
STORESAL.LIS
STOREDEG.LIS

DTR>

16.3 Modifying Records

You must first ready a domain with either modify or write access before you can modify any records it contains. Use the shared option if you want to let other users store, erase, or modify records in the domain at the same time you are accessing it.

There are three methods you can use to modify records. You can modify:

- A selected record in a collection
- All of the records in the CURRENT collection
- All of the records in an RSE

Using any of these methods, you can specify the fields you want to change. If you do not specify the fields to be changed, DATATRIEVE prompts you for all the record fields.

Note

Be very careful when modifying record fields pulled from more than one domain. This situation can exist when you are modifying records in a view based on more than one domain or in an RSE containing a CROSS clause. In these cases, if the field you are changing is stored in more than one data file, you are updating only one of those files for each field value you enter. In the sample personnel system used in this book, the domains are set up to minimize duplicate fields. If, however, you are modifying a field that needs to be changed in nine domains, you cannot escape entering the change nine times without some fairly complex statements.

Example 16-5 illustrates modifying records using a collection.

Example 16-5: Modifying Records by First Creating a Collection

```
DTR> ! Change the value of the field CONTACT_NAME in one record
DTR> ! from the COLLEGES domain.
DTR> !
DTR> READY COLLEGES MODIFY
DTR> FIND COLLEGES WITH CONTACT_NAME CONTAINING "LYNCH"
[2 records found]
DTR> LIST ALL
```

```
COLLEGE_CODE : QUIN
COLLEGE_NAME : Quinnipiac College
CONTACT_NAME : George C. Lynch
ADDRESS_DATA :
STREET      :
TOWN        : Hamden
STATE       : NH
ZIP         : 06152
```

```
COLLEGE_CODE : STAN
COLLEGE_NAME : Stanford Univ.
CONTACT_NAME : Carol Lynch
ADDRESS_DATA :
STREET      :
TOWN        : Stanford
STATE       : CA
ZIP         :
```

(continued on next page)


```

DTR> !
DTR> ! Select the record to be modified from the collection.
DTR> !
DTR> SELECT 1
DTR> !
DTR> ! If you want to be prompted to enter a value for every field
DTR> ! in the record, simply enter the keyword MODIFY. If you
DTR> ! want to be prompted to enter values only for specific fields,
DTR> ! follow the keyword MODIFY with the names of those fields.
DTR> ! Remember to include a comma between field names if there is
DTR> ! more than one of them.
DTR> !
DTR> MODIFY CONTACT_NAME
Enter CONTACT_NAME: Hayward C. Dublin
DTR> LIST

```

```

COLLEGE_CODE : QUIN
COLLEGE_NAME : Quinnipiac College
CONTACT_NAME : Hayward C. Dublin
ADDRESS_DATA :
STREET      :
TOWN        : Hamden
STATE       : NH
ZIP         : 06152

```

```

DTR> !
DTR> ! Create a collection in which all records should have the
DTR> ! same values for a field. In the following example, two
DTR> ! JOB_HISTORY records have missing values in the SUPERVISOR_ID
DTR> ! field. Both records are for employees who have the same
DTR> ! supervisor, and you want to enter one value for DATATRIEVE
DTR> ! to store in both records.
DTR> !
DTR> FIND JOB_HISTORY WITH
CON> DEPARTMENT_CODE = "ELEL" AND JOB_CODE = "EENG" AND
CON> JOB_END MISSING
[2 records found]
DTR> PRINT ALL

```

EMPLOYEE ID	JOB CODE	JOB START	JOB END	DEPARTMENT CODE	SUPERVISOR ID
00238	EENG	2-Feb-1982		ELEL	
00428	EENG	10-Jan-1982		ELEL	

```

DTR> !
DTR> ! In this case, enter MODIFY ALL followed by the field name or
DTR> ! names you want to change. DATATRIEVE prompts you once for
DTR> ! each field you specify and changes all the records in the
DTR> ! collection.
DTR> !
DTR> MODIFY ALL SUPERVISOR_ID
Enter SUPERVISOR_ID: 00356
DTR> PRINT ALL

```

EMPLOYEE ID	JOB CODE	JOB START	JOB END	DEPARTMENT CODE	SUPERVISOR ID
00238	EENG	2-Feb-1982		ELEL	00356
00428	EENG	10-Jan-1982		ELEL	00356

When you are modifying records in collections, you must be careful when using the keyword ALL. As shown in Example 16-5, you enter MODIFY ALL *only* when you want all the records in the collection to contain identical values in one or more fields. If you include an RSE in a MODIFY statement (MODIFY ... OF EMPLOYEES, for example), you get the same results for that RSE as you do for collections with MODIFY ALL—you are asking DATATRIEVE to store identical values in all the records.

If you want to put *different* field values in each collection record (without selecting each record in turn), you must set up a FOR loop. In the following sample statement, the keyword PRINT allows you to look at each record before and after you enter changes. Depending on what you want to display and change, you can specify field names following the keywords PRINT and MODIFY:

```
FOR CURRENT
  BEGIN
    PRINT
    MODIFY
    PRINT
  END
```

Example 16-6 illustrates two modify operations. In both cases, the RSE specifies records directly from a domain rather than a collection. The first operation modifies records in a FOR statement RSE. The second operation modifies records in a MODIFY statement RSE using values contained in an update file. The second operation includes the optional USING and VERIFY USING clauses you usually want to add to a MODIFY statement when designing applications for multiple users.

Example 16-6: Modifying Records in a FOR Statement RSE

```
DTR> SHOW CHANGE_SUPERS
!
! This procedure allows a user to change supervisor IDs for one
! or more current records in JOB_HISTORY. The user is prompted
! to enter the outdated supervisor ID and the department code.
! DATATRIEVE then displays a record, prompts the user to enter
! a new supervisor ID, and displays the changed record. The user
! is returned to the DTR> prompt if DATATRIEVE cannot find any
! records that meet the requirements in the FOR statement RSE.
!
```

(continued on next page)

```

PROCEDURE CHANGE_SUPERS
READY JOB_HISTORY SHARED MODIFY
FOR JOB_HISTORY WITH SUPERVISOR_ID = *."old supervisor" AND
  DEPARTMENT_CODE = *."department code" AND JOB_END MISSING
BEGIN
  PRINT EMPLOYEE_ID, EMPLOYEE_ID VIA WHO_IS_IT, SUPERVISOR_ID
  MODIFY USING SUPERVISOR_ID = *."new supervisor"
  PRINT EMPLOYEE_ID, EMPLOYEE_ID VIA WHO_IS_IT, SUPERVISOR_ID
  PRINT SKIP
END
FINISH JOB_HISTORY
END_PROCEDURE

```

```

DTR) :CHANGE_SUPERS
Enter department code: SALE
Enter old supervisor: 00200

```

EMPLOYEE ID	EMPLOYEE NAME	SUPERVISOR ID
00208	Sciacca Joe V	00200

Enter new supervisor: 00504

EMPLOYEE ID	EMPLOYEE NAME	SUPERVISOR ID
00208	Sciacca Joe V	00504
00233	Mathias Susan N	00200

Enter new supervisor: 00497

00233	Mathias Susan N	00497
.	.	.
.	.	.
.	.	.

```

DTR) !
DTR) ! Here is a more complex procedure.
DTR) !
DTR) SHOW MODIFY_EMPLOYEES
!
! This procedure uses records in an update file (associated with
! the domain TRANS_EMP) to modify records in a master file
! (associated with the domain EMPLOYEES).
!
PROCEDURE MODIFY_EMPLOYEES
SET ABORT
READY EMPLOYEES SHARED MODIFY, TRANS_EMP READ
!
! The FOR statement specifies the update file records and
! identifies them as A. The MODIFY statement specifies the
! master file records and identifies them as B. The identifiers
! A and B, when appended to field names, make it clear where
! each field value is located.
!
FOR A IN TRANS_EMP
MODIFY B IN EMPLOYEES WITH B.EMPLOYEE_ID = A.EMPLOYEE_ID USING
BEGIN

```

```

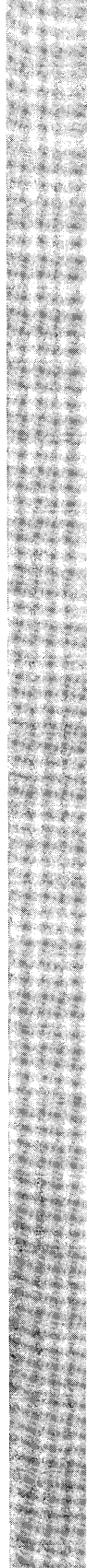
!
! You cannot change values in any key fields that do not
! allow changes. In the EMPLOYEES domain, EMPLOYEE_ID and
! SOCIAL_SECURITY are such fields. The following set of
! statements transfer from the update file record only
! values for fields that can be changed in the master file.
!
B.LAST_NAME = A.LAST_NAME
B.FIRST_NAME = A.FIRST_NAME
B.MIDDLE_INITIAL = A.MIDDLE_INITIAL
B.ADDRESS_DATA = A.ADDRESS_DATA
B.STREET = A.STREET
B.TOWN = A.TOWN
B.STATE = A.STATE
B.ZIP = A.ZIP
B.SEX = A.SEX
BRTHDAY = ARTHDAY
END VERIFY USING
!
! The VERIFY clause prints on the system line printer a
! record of the changes made to EMPLOYEES when this procedure
! executes. Because the VERIFY clause executes before each
! EMPLOYEES record is changed, it can print out the date and
! time just before the change and the EMPLOYEES record before
! the change. This is useful information to have in case
! the system fails during the procedure or if you later
! receive information that an update record is wrong.
!
BEGIN
ON LP:
  BEGIN
    DECLARE DATE_OF_TRANSACTION USAGE DATE.
    !
    ! "NOW" stores current date and time in the variable.
    ! LIST...USING X(23) displays date and time. (By default,
    ! DATATRIEVE displays date fields using X(11) which
    ! truncates the value for time.)
    !
    DATE_OF_TRANSACTION = "NOW"
    LIST DATE_OF_TRANSACTION USING X(23), SKIP
    PRINT "EMPLOYEE record:"
    !
    ! EMPLOYEES_REC is the top-level field in the record
    ! definition. In this case, it has the same name as
    ! the record definition itself. The top-level field you
    ! are working with may not. List requires a field name
    ! in this context.
    !
    LIST EMPLOYEES_REC
    PRINT SKIP
    PRINT "TRANS_EMP record:"
    LIST TRANS_EMP_REC
    PRINT SKIP 2
  END
END
END
FINISH EMPLOYEES, TRANS_EMP
END_PROCEDURE
DTR)

```



Part V
Programming with DATATRIEVE





Using Procedures and Compound Statements 17

This chapter tells you how to create and use procedures and compound statements.

A procedure is a fixed sequence of DATATRIEVE commands and statements you create, name, and store in the Common Data Dictionary (CDD). Procedures are useful when you plan to execute a series of DATATRIEVE commands and statements frequently. By putting these in a procedure, you can simply execute the procedure to save yourself much time.

Procedures are also useful when you are creating applications to help users not proficient in DATATRIEVE. You can create a procedure to carry out what they want to do, and all they have to do is execute it.

Compound statements can control a variety of execution conditions that simple statements cannot. They are particularly useful in procedures because of their flexibility.

17.1 Creating and Executing Procedures

You create a procedure with the DEFINE PROCEDURE command. You execute a procedure by entering EXECUTE or a colon (:) followed by the procedure name. If you execute a procedure using a colon, do not type a space between the colon and the procedure name.

A procedure name cannot duplicate the name of any other definition in the dictionary directory where it is stored. Like other CDD names, it can consist of from 1 to 31 letters, numbers, dollar signs, underscores, and hyphens. It must begin with a letter and end with a letter or number.

Example 17-1 creates and executes a very simple procedure called NEAT_NAME that makes a name easier to read and reduces the amount of space it occupies in a display line. The example also shows and executes a slightly more complex procedure called MAILING_LABELS.

Example 17-1: Creating a DATATRIEVE Procedure

```

DTR> ! Start a procedure definition with the keywords DEFINE
DTR> ! PROCEDURE and type the name you want for the procedure. Then
DTR> ! enter any statements and commands you want in the procedure
DTR> ! just as you would enter them interactively. When you
DTR> ! continue a statement or command over more than one line, be
DTR> ! careful that the last word or character of each line to be
DTR> ! continued indicates that the statement or command is
DTR> ! incomplete. Terminate the procedure definition with the
DTR> ! keyword END_PROCEDURE.
DTR> !
DTR> DEFINE PROCEDURE NEAT_NAME
DFN> !
DFN> ! Comment lines like these can be stored in the procedure
DFN> ! definition to make it easy to understand and read.
DFN> ! DATATRIEVE ignores any input that follows an exclamation
DFN> ! point (!) on a line. You cannot end a comment line with
DFN> ! a hyphen, however.
DFN> !
DFN> ! The vertical bars (|or|) in the following variable
DFN> ! definition suppress leading and trailing spaces that occur in
DFN> ! field values. Chapter 19 tells you more about using them.
DFN> !
DFN> DECLARE NEAT_NAME COMPUTED BY
DFN>   FIRST_NAME!! " |MIDDLE_INITIAL!!" " |LAST_NAME
DFN>   QUERY_HEADER IS "NAME".
DFN> END_PROCEDURE
DTR> SHOW PROCEDURES
Procedures:
      CHANGE_SUPERS;1 DATESTORE;1      DATESTORE2;1  DATESTORE3;1
      DATESTORE4;1  DATESTORE5;1  MODIFY_AN_EMPLOYEE_RECORD;7
      MAILING_LIST;2                                MODIFY_EMPLOYEES;11
NEAT_NAME;1
              NEW_PEOPLE;4              WASTE_BASKET;1

DTR> !
DTR> ! ADDRESS_LIST is a view that contains names and addresses of
DTR> ! current employees.
DTR> !
DTR> READY ADDRESS_LIST
DTR> :NEAT_NAME
DTR> PRINT NEAT_NAME, TOWN, STATE OF FIRST 5 ADDRESS_LIST

```

NAME	TOWN	STATE
Alvin A Toliver	Chocorua	NH
Terry D Smith	Chocorua	NH
Rick Dietrich	Boscawen	NH
Janet Kilpatrick	Marlow	NH
Norman Nash	Meadows	NH

```
DTR> !
DTR> ! MAILING_LABELS is a procedure that prints out the names
DTR> ! and addresses of employees as they would appear on a
DTR> ! label for postal service distribution. The COL expressions
DTR> ! determine the line position on which the following print
DTR> ! element begins. The hyphen in parentheses (-) specifies
DTR> ! that the preceding field is displayed without a header.
DTR> !
```

```
DTR> SHOW MAILING_LABELS
PROCEDURE MAILING_LABELS
READY ADDRESS_LIST
FOR ADDRESS_LIST
    PRINT COL 3, FIRST_NAME!!" "MIDDLE_INITIAL!!" "LAST_NAME,
        COL 3, ADDRESS_DATA (-),
        COL 3, STREET (-),
        COL 3, TOWN!!", "STATE,
        COL 18, ZIP (-),
        SKIP 2
FINISH ADDRESS_LIST
END_PROCEDURE
```

```
DTR> :MAILING_LABELS
Alvin A Toliver
RFD 5
146 Parnell Place
Chocorua, NH
                                03817

Terry D Smith

120 Tenby Dr.
Chocorua, NH
                                03817

Rick Dietrich
Apartment 45
19 Union Square
Boscawen, NH
                                03301
.      .      .
.      .      .
.      .      .
```

```
DTR>
```

17.2 Correcting and Changing Procedures

When you enter `END_PROCEDURE`, `DATATRIEVE` checks your definition for some of the syntax errors you might have made. It checks for other syntax errors at the time you execute the procedure. If `DATATRIEVE` finds a syntax error, it displays an error message and does not store the definition. If this happens, immediately type `EDIT` and press the `RETURN` key. `DATATRIEVE` puts your entire `DEFINE` command in the edit buffer. You must look over your statements to determine which one produced the error, correct your mistake, and exit the editor. You may have to repeat this process several times to take care of all the syntax mistakes in your procedure definition.

It is wise, therefore, to create long procedures in stages. In the first stage, put only the first few statements and commands you eventually want in your procedure before typing `END_PROCEDURE`. If `DATATRIEVE` displays an error message, you have only a few lines to troubleshoot when you type `EDIT`. If `DATATRIEVE` returns you to the `DTR>` prompt, `DATATRIEVE` stored the first version of your procedure definition. You then type `EDIT` followed by the procedure name to add more to your procedure.

Note

The `EDIT` entry immediately after the first version is stored should be followed by the procedure name. Doing this ensures that a `REDEFINE` command appears at the top of the definition copied to your edit buffer. If you simply type `EDIT` and press the `RETURN` key, you get your last command. Immediately after a procedure is stored, your last command is `DEFINE PROCEDURE`, not `REDEFINE PROCEDURE`. Any `DEFINE` command produces an error when the named object already exists in the dictionary directory. If your last command was `REDEFINE PROCEDURE`, entering the keyword `EDIT` by itself causes no problems.

Creating procedures in stages can save you the frustration of searching through 50 lines of input to find a missing quotation mark or period. It can, however, create quite a few versions of your procedure definition before you are done. Chapter 7 tells you how to get rid of the earlier versions.

17.3 Using Compound Statements

A compound statement can contain one or more subordinate statements, but never any commands. If you begin a statement with any of the following keywords, DATATRIEVE recognizes it as a compound statement:

- REPEAT
- FOR
- BEGIN
- IF
- CHOICE
- WHILE

When you use the keyword THEN to join two statements, you also create a compound statement.

You use compound statements to process one or more records in an RSE, to set up conditions to control the number of times something is done, or to set up conditions to specify whether something is done at all.

Here are some templates that illustrate a few compound statements useful in procedures. By substituting a phrase for a real statement, the templates make it easier for you to focus on which statements are subordinate to others. These templates do not illustrate all the options you have when putting statements together but they give a beginner some useful “roadmaps” to follow:

```
REPEAT this-many-times STORE...
```

```
FOR each-each-of-the-records-in-this-RSE  
  Do-this... THEN Do-this...
```

```
STORE in-this-domain USING  
  BEGIN  
    Prompt-for-a-field-value  
    Prompt-for-a-field-value  
    Prompt-for-a-field-value
```

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

```
END    VERIFY USING  
      BEGIN  
        Maybe-check-a-field-entry  
        Maybe-print-a-message  
        Maybe-get-a-new-field-value  
      END
```

(continued on next page)

```

FOR each-of-the-records-in-this-RSE
  MODIFY USING
    BEGIN
      Print-the-record-before-the-change
      Maybe-print-a-message
      Prompt-for-a-field-value
      Prompt-for-a-field-value
      Prompt-for-a-field-value
      . . .
      . . .
      . . .
      Print-the-record-after-the-change
    END
  VERIFY USING
    BEGIN
      Maybe-print-a-message
      Maybe-check-a-field-entry
      Maybe-reprompt-for-a-field-value
      Maybe-print-auditing-information
    END

```

```

WHILE this-condition-is-true
  BEGIN
    Do-this...
    And-also...
    . . .
    . . .
    . . .
    And-finally...
  END

```

```

IF this-condition-is-true THEN
  BEGIN
    Do-this...
    And-also...
    . . .
    . . .
    . . .
    And-finally...
  END
ELSE
  Do-this...

```

CHOICE

```
FIELD1 EQUAL FIELD2 THEN Do-this...  
FIELD1 LESS FIELD2 THEN Do-this...  
FIELD1 GREATER FIELD2 THEN Do-this...
```

```
ELSE Do-this...  
END__CHOICE
```

As shown in the templates, compound statements often contain other compound statements. When you create your own procedures, you often find that you start out with a word like **FOR** or **WHILE** and type many lines before you end that statement. The templates use indentation to show which statements are contained in others. It is a good idea for you to do this, too. **DATATRIEVE** does not require indentation but you will find that compound statements are easier to read if you include it.

The following sections discuss each type of compound statement.

17.3.1 Combining Statements with the REPEAT Statement

The **REPEAT** statement causes **DATATRIEVE** to execute the next statement a specified number of times. In response to the following statement, **DATATRIEVE** prompts you to enter field values for each of five records:

```
REPEAT 5 STORE EMPLOYEES
```

The number of times **DATATRIEVE** executes the subordinate statement can be specified as an expression rather than an integer; for example, **REPEAT (FIELD1 * 4)**. If you use an expression, it must result in a positive whole number when **DATATRIEVE** evaluates it.

17.3.2 Combining Statements with the FOR Statement

A **FOR** statement causes **DATATRIEVE** to execute the next statement on each of the records in an **RSE**. The following **FOR** statement causes **DATATRIEVE** to print each of the records in **DEGREES** that contain **PRDU** in the **COLLEGE_CODE** field:

```
FOR DEGREES WITH COLLEGE_CODE = "PRDU"  
PRINT
```

17.3.3 Combining Statements with the Keyword THEN

The keyword THEN joins two statements. THEN is most useful when you have two or three simple operations to perform in a loop:

```
FOR JOB_HISTORY WITH EMPLOYEE_ID = "00205"  
  PRINT THEN MODIFY THEN PRINT
```

Using THEN to join long statements makes them difficult to read. A BEGIN-END statement is a good substitute for repeated THEN statements when you want to execute several long statements.

17.3.4 Combining Statements in a BEGIN-END Block

A BEGIN-END statement (also called a BEGIN-END block) causes DATATRIEVE to treat several statements as one statement. It is especially useful within FOR, MODIFY, STORE, and REPEAT statements. A BEGIN-END block defines the set of statements that must execute for each record in an RSE or each time a condition is true:

```
FOR EMPLOYEES WITH EMPLOYEE_ID = *."employee number"  
  BEGIN  
    LIST EMPLOYEE_ID, EMPLOYEE_NAME, EMPLOYEE_ADDRESS  
    MODIFY EMPLOYEE_NAME, EMPLOYEE_ADDRESS  
    LIST EMPLOYEE_ID, EMPLOYEE_NAME, EMPLOYEE_ADDRESS  
  END
```

A BEGIN-END block inside a REPEAT statement causes DATATRIEVE to repeat an entire sequence of statements. If you want to store numerous records using selected fields, you can use a BEGIN-END block to repeat the sequence of prompting statements:

```
REPEAT 20 STORE JOBS USING  
  BEGIN  
    JOB_CODE = *.JOB_CODE  
    JOB_TITLE = *.JOB_TITLE  
  END
```

17.3.5 Combining Statements with the WHILE Statement

The WHILE statement tells DATATRIEVE to repeat the subordinate statement as long as a specified condition is true. You can use a WHILE statement to control the number of times DATATRIEVE loops through a PRINT, MODIFY, or STORE operation. In the following example, MORE_RECORDS is a variable whose value is under the control of the person storing records in EMPLOYEES:

```
DECLARE MORE_RECORDS PIC X DEFAULT VALUE IS "Y".
READY EMPLOYEES SHARED WRITE
WHILE MORE_RECORDS CONTAINING "Y"
  BEGIN
    STORE EMPLOYEES
    MORE_RECORDS = *."Y if you are storing more records, N if not"
  END
```

The condition specified in the WHILE statement takes the form of a Boolean (relational) expression. In the last example, MORE_RECORDS CONTAINING "Y" is a Boolean expression. In the following example, the WHILE statement contains the Boolean expression NUM LE 10 (LE is one way to write "less than or equal to"):

```
DTR> DECLARE NUM PIC 99.
DTR> DECLARE ITS_SQUARE COMPUTED BY NUM * NUM.
DTR> WHILE NUM LE 10
CON>   BEGIN
CON>     PRINT NUM, ITS_SQUARE
CON>     NUM = NUM + 1
CON>   END
```

NUM	ITS SQUARE
00	0
01	1
02	4
03	9
04	16
05	25
06	36
07	49
08	64
09	81
10	100

```
DTR>
```

Note that MORE_RECORDS and NUM are both *variable* field names. You cannot specify a *record* field name on the left side of a Boolean expression when setting up a condition in a WHILE statement. If you need to put a record field name in that position, you must prompt the user to enter one (WHILE *."field name" ...).

Chapter 18 tells you more about creating simple and complex Boolean expressions and using variables.

17.3.6 Combining Statements with the IF-THEN and IF-THEN-ELSE Statements

An IF-THEN statement causes DATATRIEVE to execute a statement *only* when a condition is true. The IF component of the statement contains a Boolean expression. The THEN component contains the simple or compound statement to be executed when the expression is true. If the expression is false, DATATRIEVE goes on to process the next statement if any exists.

An IF-THEN-ELSE statement causes DATATRIEVE to execute *one of two* statements depending on whether a condition is true or not. The THEN component contains the simple or compound statement to be executed when the expression is true. The ELSE component specifies the simple or compound statement to be executed when the expression is false.

When you combine statements with IF-THEN-ELSE, you can omit the keyword THEN because DATATRIEVE knows an IF component is always incomplete. When you include an ELSE component, however, you cannot omit the keyword ELSE or put it on a line following the THEN statement. Type ELSE on the same line as the last part of the THEN statement. This is how you tell DATATRIEVE that you are not entering a simple IF-THEN statement.

Here is one example of an IF-THEN-ELSE statement:

```
IF REVIEW_DATE < CUT_OFF_DATE
THEN PRINT EMPLOYEE_ID, EMPLOYEE_ID VIA WHO_IS_IT,
      REVIEW_DATE, " Needs a review" ELSE
PRINT EMPLOYEE_ID, EMPLOYEE_ID VIA WHO_IS_IT,
      REVIEW_DATE, " Review up-to-date"
```

Here is the way the statement can be included in a procedure:

```
DTR> SHOW REVIEW_DATES
PROCEDURE REVIEW_DATES
READY JOB_HISTORY
!
DECLARE CUT_OFF_DATE USAGE DATE.
CUT_OFF_DATE = *."date six months ago"
!
FOR JOB_HISTORY WITH SUPERVISOR_ID = *."supervisor ID" AND
      JOB_END MISSING SORTED BY REVIEW_DATE
!
  IF REVIEW_DATE < CUT_OFF_DATE
  THEN PRINT EMPLOYEE_ID, EMPLOYEE_ID VIA WHO_IS_IT,
        REVIEW_DATE, " Needs a review" ELSE
  PRINT EMPLOYEE_ID, EMPLOYEE_ID VIA WHO_IS_IT,
        REVIEW_DATE, " Review up-to-date"
```

```
!
FINISH JOB_HISTORY
RELEASE WHO_IS_IT, CUT_OFF_DATE
END_PROCEDURE
```

```
DTR> :REVIEW_DATES
Enter date six months ago: 1/9/83
Enter supervisor ID: 00267
```

EMPLOYEE ID	EMPLOYEE NAME	REVIEW DATE	
00497	Weist Robert	30-Sep-1982	Needs a review
00419	Clarke Aruwa Q	15-Dec-1982	Needs a review
00355	Gutierrez Joe	1-Jan-1983	Needs a review
00184	Frydman Louie T	5-Jan-1983	Needs a review

EMPLOYEE ID	EMPLOYEE NAME	REVIEW DATE	
00464	Aaron Alvin	9-Jan-1983	Review up-to-date
00216	Lobdell Arleen Y	13-Apr-1983	Review up-to-date
00244	Boyd Ann B	24-Apr-1983	Review up-to-date
00283	Dallas Paul	4-May-1983	Review up-to-date
00200	Ziemke Al F	21-May-1983	Review up-to-date
00501	Gramby Terry	7-Jun-1983	Review up-to-date
00241	Keisling Edward	3-Jul-1983	Review up-to-date

```
DTR>
```

17.3.7 Combining Statements with the CHOICE Statement

A CHOICE statement causes DATATRIEVE to execute one of a series of statements depending on the evaluation of a Boolean expression associated with each statement.

The CHOICE statement is a good substitute for nested IF-THEN-ELSE statements:

```
IF A > B THEN PRINT "A's bigger" ELSE
  IF A = B THEN PRINT "A's the same size" ELSE
    IF A < B THEN PRINT "A's smaller" ELSE
      PRINT "A's a strange duck"
```

```
CHOICE
  A > B THEN PRINT "A's bigger"
  A = B THEN PRINT "A's the same size"
  A < B THEN PRINT "A's smaller"
  ELSE PRINT "A's a strange duck"
END_CHOICE
```

The ELSE clause is optional in a CHOICE statement. You can also omit the keyword THEN, although doing so makes the statement more difficult to read.

Do not add the keyword IF before a Boolean expression in a CHOICE statement, even though you might say "if" to yourself as you read it.

Here is an example of a CHOICE statement in a procedure:

```
DTR> SHOW REVIEW_DATES_TWO
PROCEDURE REVIEW_DATES_TWO
READY JOB_HISTORY
!
DECLARE CURRENT_DATE USAGE DATE.
CURRENT_DATE = "TODAY"
!
FOR JOB_HISTORY WITH SUPERVISOR_ID = "00267" AND
                JOB_END MISSING SORTED BY REVIEW_DATE
CHOICE
!
! GE 210 days is 7 months or more...
!
CURRENT_DATE - REVIEW_DATE GE 210 THEN
    PRINT EMPLOYEE_ID, REVIEW_DATE,
    "At least one month overdue"
!
! BT 209 AND 180 days is between 7 and 6 months...
!
CURRENT_DATE - REVIEW_DATE BT 209 AND 180 THEN
    PRINT EMPLOYEE_ID, REVIEW_DATE,
    "A few weeks overdue"
!
! BT 179 AND 150 days is between 6 and 5 months...
!
CURRENT_DATE - REVIEW_DATE BT 179 AND 150 THEN
    PRINT EMPLOYEE_ID, REVIEW_DATE,
    "Not overdue, but schedule"
!
! No one else needs a performance review...
!
ELSE PRINT EMPLOYEE_ID, REVIEW_DATE,
    "Reviewed recently"
END_CHOICE
!
FINISH JOB_HISTORY
END_PROCEDURE

DTR>
```

Chapter 18 tells you more about manipulating date fields.

17.4 Guidelines for Writing Procedures and Compound Statements

This section discusses some restrictions and error conditions that you should know about before you begin to write your own procedures.

17.4.1 Using FIND, SELECT, SORT, REDUCE, and DROP Statements

You can include FIND, SELECT, SORT, REDUCE, or DROP statements in procedures but include them only as simple statements. Do not enter these statements as part of a compound statement.

In addition, remember that when a FIND statement is in a procedure, DATATRIEVE does not display a message to tell you how many records it found for the collection. If DATATRIEVE did not find any records to meet the specifications in the FIND statement RSE, a subsequent SELECT statement will produce an error message.

17.4.2 Avoiding Looping Mistakes

You can create an **infinite loop** (looping that does not stop) by setting up a condition that is always true. The following is an obvious example of a condition that is always true (The symbol < means less than):

```
WHILE 0 < 1
  PRINT "This is an infinite loop. Enter CTRL/C to stop it."
```

You can create the same sort of situation when you use expressions that contain variables whose values you do not control. Refer back to the examples for the WHILE statement earlier in this chapter. If the values of the condition variables (MORE_RECORDS and NUM) were changed outside the BEGIN-END block subordinate to the WHILE statement, an infinite loop would result.

You can invoke a procedure within a loop but you should put the EXECUTE (:) statement inside a BEGIN-END block when you do. Otherwise, DATATRIEVE may execute only the first statement or command in the procedure:

```
FOR rse
  BEGIN
    :procedure-name
  END
```

```
REPEAT n-times
  BEGIN
    :procedure-name
  END
```

You can let procedures execute other procedures but do not let a procedure invoke itself, either directly or indirectly. If you do, you might create an infinite loop.

17.4.3 Invoking DATATRIEVE Procedures from VMS Command Files

You might want to invoke a DATATRIEVE procedure from a VMS command file. If you do this, you invoke the procedure with the keyword EXECUTE rather than the colon (:). In the following example, DTR32 is a DCL symbol:

```
$ TYPE STOREEMP.COM
$ ! STOREEMP.COM executes a DATATRIEVE procedure to
$ ! store new employee records and then prints the
$ ! auditing information created by the procedure.
$ !
$ DTR32 EXECUTE CDD$TOP.PERSONNEL.STORE_EMPLOYEES
$ PRINT/QUEUE=SYS$PRINT AUDIT.LOG
$ @STOREEMP.COM
      .      .      .
      .      .      .
      .      .      .
$
```

17.4.4 Controlling Execution on Error Conditions

You can use the SET ABORT or SET NO ABORT commands to control what happens if DATATRIEVE encounters an error condition when processing a procedure. SET ABORT is the normal default when you invoke DATATRIEVE.

When SET ABORT is in effect, DATATRIEVE exits a procedure when either of the following conditions is true:

- It finds an error condition when trying to execute one of the statements or commands in the procedure.
- It processes a CTRL/Z or CTRL/C entered by the person who invokes the procedure.

If either of these conditions is true and SET NO ABORT is in effect, DATATRIEVE does not exit the procedure. It does abort the statement it is executing but continues processing any remaining statements and commands.

Generally, you want SET ABORT in effect when your procedure readies domains that contain the data you want to access. If those domains cannot be readied, there is no point in continuing with the rest of the procedure. If SET NO ABORT were in effect, DATATRIEVE would produce error messages as it processed any statements referring to the domains.

On the other hand, SET NO ABORT should be in effect if your procedure prompts users to enter values for more than one record. In this case, you can expect someone to enter CTRL/Z once in a while to keep an entry for one of the records from being stored. You want users to reenter the looping cycle so they can store or change more records. If SET ABORT is in effect, DATATRIEVE aborts the remainder of a procedure or command file when you enter CTRL/Z.

The ABORT statement lets you specify an error condition appropriate for the operation you are performing. The ABORT statement terminates execution of the compound statement or statements containing it and can print a message explaining the termination.

An IF or CHOICE statement sets up the condition under which the ABORT statement executes. In the following example, the ABORT statement is in the VERIFY clause of a STORE statement. It executes when a value for EMPLOYEE_ID already exists in the EMPLOYEES domain. Note that the ABORT statement terminates all statements to which it is subordinate. In this case, the user does not get to store any more records. Because SET NO ABORT is in effect, however, the FINISH command and the final PRINT statement do execute:

```
DTR> SHOW STORE_EMPLOYEE
PROCEDURE STORE_EMPLOYEE
SET ABORT
READY EMPLOYEES SHARED WRITE
SET NO ABORT
DECLARE MORE_RECORDS PIC X.
MORE_RECORDS = "Y"
WHILE MORE_RECORDS CONT "Y"
  BEGIN
    STORE EMPLOYEES VERIFY USING
      IF EMPLOYEE_ID IN WHO_IS_IT THEN
        ABORT "Employee number '!EMPLOYEE_ID!'" already exists"
        MORE_RECORDS = *."Y if storing more records, N if not"
      END
  FINISH EMPLOYEES
  PRINT "Exit from procedure STORE_EMPLOYEE."
END_PROCEDURE
```

(continued on next page)

```

DTR> :STORE_EMPLOYEE
Enter EMPLOYEE_ID: 00799
Enter LAST_NAME: TORRELL
Enter FIRST_NAME: MAY
.
.
.
Enter Y if storing more records, N if not: Y
Enter EMPLOYEE_ID: 00164
Enter LAST_NAME: CHARLES
Enter FIRST_NAME: LISA
.
.
.
ABORT: Employee number 00164 already exists
Exit from procedure STORE_EMPLOYEE.

DTR>

```

Statements to control and specify error conditions can be numerous and complex in large scale applications. This section is designed only to introduce you to the topic. There are restrictions that apply to the ABORT statement that are not listed here. If you want more detailed information on handling error conditions, you should read the sections on the SET command and the ABORT statement in the *VAX DATATRIEVE Reference Manual*.

17.5 Getting a Procedure to Work the Way You Want

When you execute a procedure for the first time, you might find one of two things happens:

- DATATRIEVE uncovers some syntax problems that it did not catch at the time you stored the procedure.
- The results produced by the procedure are not what you want.

The SET VERIFY and OPEN commands are useful when you are troubleshooting problems with procedures.

17.5.1 Displaying Command File and Procedure Input During Execution

SET VERIFY displays input from VMS command files and the editing buffer as they are processed. SET NO VERIFY turns off this input display. (The SHOW SET_UP command tells you whether SET VERIFY or SET NO VERIFY is in effect during your DATATRIEVE session.) SET VERIFY and SET NO VERIFY are also DCL settings. The default for your DATATRIEVE session is whichever one is in effect at DCL level when you invoke DATATRIEVE.

SET VERIFY does not display statements and commands from DATATRIEVE procedures as they are processed. It is easy, however, to recreate your DATATRIEVE procedure as a VMS command file so you can take advantage of SET VERIFY:

1. Write your procedure definition to a command file using the format:

```
EXTRACT procedure-name ON filename.COM
```
2. Exit DATATRIEVE and edit the command file to remove the following lines:

```
DELETE procedure-name;  
REDEFINE procedure-name  
END__PROCEDURE
```

If you execute the procedure in a compound statement (FOR rse :procedure-name, for example), you must also include the compound statement in the command file.

3. Invoke DATATRIEVE again and execute the command file:

```
@filename
```

If SET VERIFY is in effect when you execute your command file, DATATRIEVE displays input from the command file as it is processed. If there is a syntax error, you can see exactly where it occurs. Check your incorrect input for typographical errors. Then check the final word or character of the preceding input line. (A common mistake is to enter parts of statements or commands so that DATATRIEVE cannot recognize that you are continuing them on the next line.) If you do not recognize your syntax mistake, you might have to check the format in the *VAX DATATRIEVE Reference Manual* or the *VAX DATATRIEVE Pocket Guide* that applies to the statement, command, clause, or expression in which the error occurred.

If parts of your command file do not execute when you expect them to (DATATRIEVE apparently “skips” over some parts of a compound statement or loops too many or too few times), SET VERIFY allows you to see which lines are being executed before the skip occurs and where DATATRIEVE starts processing again. You can then inspect the input at the point where the problem seems to start.

When you are finished editing the command file and it works the way you want, you can edit it one more time to insert:

1. REDEFINE procedure-name as the first line of the file
2. END__PROCEDURE as the last line of the file

At DATATRIEVE command level, you can then execute the command file a final time to create it as a new version of your DATATRIEVE procedure.

You might ask at this point, “Why not use command files rather than procedures?” You can. However, there are disadvantages to using command files. For example:

- You cannot execute command files inside a compound statement, such as a FOR statement
- You lose advantages such as history and access control lists that are available for CDD objects

17.5.2 Writing a Session Log to a File

You can use the OPEN command to create a log file of your DATATRIEVE session in your VMS directory. After you execute a command file, you can close this file with a close command and exit DATATRIEVE to print it. This is useful when you are troubleshooting a long command file or one that produces much output. In this case, your screen often does not accommodate all the lines you need to see at one time. In the following example, PROBLEMS.LOG is the log file name, but you can choose any valid VMS file specification you want:

```
DTR> OPEN PROBLEMS.LOG
DTR> SET VERIFY
DTR> @MYFILE
      .
      .
      .
DTR> CLOSE          ! Do not enter the file name following CLOSE.
DTR> SET NO VERIFY
DTR> EXIT
$ PRINT/QUEUE=SYS$PRINT PROBLEMS.LOG
```

17.5.3 Checking the Last Word or Character of Input Lines

Some procedure and DATATRIEVE command file problems occur because one or more DATATRIEVE statements or commands were not properly continued from one input line to the next. Read the section in Chapter 5 on processing more than one input line as a unit for a discussion of this topic. Then try entering interactively a statement or command that does not work properly in the procedure or command file. Enter it exactly as it is written in the procedure or command file. If DATATRIEVE returns the DTR> prompt rather than a CON> prompt after you enter an input line that you plan to continue, you need to correct your input format for the statement.

Defining and Calculating Values with DATATRIEVE 18

This chapter provides details on forming and using values with DATATRIEVE. It provides more comprehensive coverage of concepts to which you have been introduced in preceding chapters.

18.1 Using DATATRIEVE Expressions

You use value expressions, Boolean expressions, and record selection expressions (RSEs) every time you tell DATATRIEVE what you want to print, find, store, or modify:

- A **value expression** consists of characters and symbols that specify a value for DATATRIEVE to use when executing statements.

PRINT EMPLOYEE_NAME, PRINT “Do not press TAB”, and PRINT TOTAL SALARY all contain value expressions to tell DATATRIEVE what to print.

- A **Boolean expression** consists of value expressions, relational operators, and Boolean operators and specifies a relationship between value expressions for DATATRIEVE to evaluate when executing statements. DATATRIEVE evaluates Boolean expressions as either true or false.

DEPARTMENT_CODE = “ADMN”, DEPARTMENT_CODE = *.”department”, and JOB_CODE = “CLRK” AND JOB_END MISSING are examples of Boolean expressions.

- A **record selection expression (RSE)** consists of a record source, Boolean expressions, and other clauses that identify the records with which you want to do something.

EMPLOYEES WITH BIRTHDAY < "1/1/1946" SORTED BY BIRTHDAY is an example of an RSE.

The clauses you can use when creating RSEs are detailed in Chapters 14 and 15 and are not discussed further in this chapter. RSEs contain Boolean expressions, however, and you can get additional ideas for creating your own RSEs from reading this chapter.

18.1.1 Value Expressions

Table 18-1 lists all the value expressions recognized by DATATRIEVE. These are all discussed later in this chapter, with the exception of the last two value expressions: table is discussed in Chapter 12 and concatenated is discussed in Chapter 19.

Table 18-1: Value Expressions

Value Expression	Example
Literal	"Order number:"
Record field name	LAST_NAME
Variable field name	RECORD_COUNTER
Prompting	*."department"
Arithmetic	PRICE * .75
Conditional	IF PRICE LE 5 THEN PRICE * .9
FORMAT	FORMAT ID USING 9(5)
Statistical	TOTAL PRICE
Date	NOW
Table	IN JOBS_TABLE
Concatenated	FIRST_NAME LAST_NAME

18.1.1.1 Literals — Literals are what most people think of first when they see the term value. A literal explicitly states what you want to store or find in a field or what you want to print. There are two types of literals:

- A character string, or text, literal is a string of up to 253 printing characters. Except when you enter them in response to a DATATRIEVE prompt, you must enclose character string literals in quotation marks. Printing characters consist of uppercase and lowercase letters, numbers, and the following special characters:

! @ # \$ % & * () - _ = + " '
 [{] } ; : \ | , < . > / ?

- A numeric literal is a string of up to 31 digits. It can also contain a decimal point (except in the rightmost position). Negative numbers can be preceded by a minus sign (-). If you use a numeric literal to assign a value to a numeric field, the PIC, USAGE, and sometimes SCALE clauses for the field can limit the number of digits you can enter for the field. These clauses also determine whether you can enter a decimal point or a sign.

The following example uses both a character string literal and a numeric literal:

```
DTR> PRINT "The number is:", 21  
The number is: 21
```

```
DTR>
```

Note that DATATRIEVE preserves the case of any letters you type for character string literals. This is also true when you retrieve values for a text field in response to a DATATRIEVE prompt. Except when you use the CONTAINING (CONT) operator to search for field values, you must type letters using the case in which they are stored in a field.

Chapter 19 shows you how to enter long character string literals over several lines. It also shows you how to append value expressions to one another when you need to create a string longer than 253 characters.

18.1.1.2 Record Field Names — When you use the name or query name of an elementary or group field as a value expression, DATATRIEVE uses whatever values are stored in that field. The following PRINT statement specifies the group field NAME as a value expression:

```
DTR> PRINT NAME OF FIRST 3 EMPLOYEES
```

LAST NAME	FIRST NAME	INIT
Toliver	Alvin	A
Smith	Terry	D
Dietrich	Rick	

```
DTR>
```

18.1.1.3 Variable Field Names — A variable is a field you define and name with the DECLARE statement. After you create the variable, you can assign a value to it and use its name as a value expression just as you can a record field name. When you create a variable, you can define it using any of the field definition clauses (except for the OCCURS and REDEFINES clauses) used to define a record field. Unless you specify a DEFAULT or MISSING VALUE clause in your variable definition, DATATRIEVE gives text variables an initial value of spaces and numeric variables an initial value of zero.

The following example creates three variables and manipulates their assigned values. Note that you end a variable field definition with a period, just as you do a field definition for a record:

```
DTR> DECLARE NUM1 PIC 99.  
DTR> PRINT NUM1
```

```
NUM1
```

```
  00
```

```
DTR> NUM1 = 15  
DTR> PRINT NUM1
```

```
NUM1
```

```
  15
```

```
DTR> DECLARE NUM2 PIC 99.  
DTR> NUM2 = 10  
DTR> PRINT NUM2
```

```
NUM2
```

```
10
```

```
DTR> PRINT NUM1 * NUM2  
150
```

```
DTR> DECLARE NUM3 COMPUTED BY NUM1 * NUM2.  
DTR> PRINT NUM3
```

```
NUM3
```

```
150
```

```
DTR>
```

NUM1, NUM2, and NUM3 are *global variables*. A global variable exists until you get rid of it with the **RELEASE** command, declare another global variable with the same name, or exit **DATATRIEVE**.

Variables declared within a **BEGIN-END** block or a **THEN** statement are *local variables*. A local variable disappears when the **BEGIN-END** block or the **THEN** statement containing it finishes executing. As the following example illustrates, the definition and value of a local variable has no effect on another local variable in an outer **BEGIN-END** block or on a global variable:

```
DTR> DECLARE X PIC X(6).  
DTR> X = "Global"  
DTR> PRINT X
```

```
X
```

```
Global
```

```
DTR> BEGIN  
CON>   DECLARE X PIC X(7).  
CON>   X = "Local 1"  
CON>   PRINT X  
CON>     BEGIN  
CON>       DECLARE X PIC X(7).  
CON>       X = "Local 2"  
CON>       PRINT X  
CON>     END  
CON>   PRINT X  
CON> END
```

(continued on next page)

```
X
Local 1
X
Local 2
X
Local 1
DTR> PRINT X
X
Global
DTR>
```

18.1.1.4 Prompting Value Expressions — To make your procedures more versatile, you can have DATATRIEVE prompt for a value to use when executing statements. Do this by using a prompting value expression rather than explicitly typing a literal, field name, or variable name in your statements. The following example shows the variations you can use when typing prompting value expressions:

```
DTR> PRINT *.name
Enter NAME: ERIN
ERIN

DTR> PRINT *."your name"
Enter your name: ERIN
ERIN

DTR> REPEAT 3 PRINT *."a greeting"
Enter a greeting: Hi
Hi
Enter a greeting: Hello
Hello
Enter a greeting: Bonjour
Bonjour

DTR> REPEAT 3 PRINT **."a greeting"
Enter a greeting: Hello
Hello
Hello
Hello

DTR>
```

As you can see, you put quotation marks around the message for the Enter prompt when you want to include lowercase letters and spaces. You use a double asterisk (**) in your prompting expression when you are creating a loop and you want DATATRIEVE to prompt for one value to repeat every time through the loop. The double asterisk prompt is useful when you are storing or modifying records in a loop and you know a field value is going to be the same for all records.

18.1.1.5 Arithmetic Expressions — An arithmetic expression is any number of numeric value expressions separated by arithmetic operators. DATATRIEVE recognizes four arithmetic operators. Table 18-2 shows each arithmetic operator and the operation it performs.

Table 18-2: Arithmetic Operators

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division

The following PRINT statement contains an elementary arithmetic expression that adds three numeric literals:

```
DTR> PRINT 4 + 5 + 1
10
DTR>
```

Arithmetic expressions can contain record or variable field names in addition to or in place of numeric literals. DATATRIEVE uses the values stored in those fields when it performs the arithmetic operation. If you are subtracting values in fields, however, be sure to include spaces around the minus sign. If you do not include spaces, DATATRIEVE interprets the minus sign as a hyphen and converts it to an underscore:

```
DTR> PRINT X-1
"X_1" is undefined or used out of context.
DTR>
```


The following three rules control the order in which DATATRIEVE performs arithmetic operations:

1. First evaluate any part of the expression in parentheses.
2. Then perform multiplications and divisions from left to right in the arithmetic expression.
3. Finally, perform additions and subtractions from left to right in the arithmetic expression.

You can control the order in which DATATRIEVE performs arithmetic operations by using parentheses. This can affect the results DATATRIEVE gets when it evaluates the expression:

```
DTR> PRINT (6 * 7) + 5
      47
```

```
DTR> PRINT 6 * (7 + 5)
      72
```

```
DTR> PRINT 12 - 6 / 2
      9.000
```

```
DTR> PRINT (12 - 6) / 2
      3.000
```

```
DTR>
```

18.1.1.6 Conditional Value Expressions — The DATATRIEVE conditional value expressions (IF-THEN-ELSE and CHOICE) look very much like the DATATRIEVE statements that use the same keywords. The conditional statements (discussed in Chapter 17) execute subordinate statements, depending on whether a condition is true or false. The conditional value expressions assign field values, depending on whether a condition is true or false.

The IF-THEN-ELSE value expression can assign one of two values. In the following example, it assigns values to a variable, DISCOUNT_PRICE, depending on the value in another variable, PRICE:

```
DTR> DECLARE PRICE PIC 9(4)V99 EDIT_STRING $$$$$.99.
DTR> DECLARE DISCOUNT_PRICE COMPUTED BY
CON>   IF PRICE LE 500 THEN PRICE * .95 ELSE PRICE * .75
CON>   EDIT_STRING $$$$$.99.
DTR> PRICE = 100.00
DTR> PRINT PRICE, DISCOUNT_PRICE
```

```

          DISCOUNT
PRICE    PRICE
$100.00 $95.00

```

```

DTR> PRICE = 1000.00
DTR> PRINT PRICE, DISCOUNT_PRICE

```

```

          DISCOUNT
PRICE    PRICE
$1000.00 $750.00

```

```
DTR>
```

The CHOICE value expression can assign one of several values. In the following example, it assigns values to a variable, NEW_SALARY, depending on the values in a record field, REVIEW_CODE:

```
DTR> PRINT CURRENT
```

NAME	REVIEW CODE	CURRENT SALARY
HARRY	1	\$51,712.00
SAM	5	\$11,676.00
HORTENSE	4	\$18,497.00
SALLY	3	\$17,510.00
JANE	2	\$32,254.00

```

DTR> DECLARE NEW_SALARY PIC 9(6)V99 COMPUTED BY
CON> CHOICE
CON> REVIEW_CODE = "1" THEN CURRENT_SALARY * 1.20
CON> REVIEW_CODE = "2" THEN CURRENT_SALARY * 1.15
CON> REVIEW_CODE = "3" THEN CURRENT_SALARY * 1.10
CON> REVIEW_CODE = "4" THEN CURRENT_SALARY
CON> ELSE CURRENT_SALARY * .90
CON> END_CHOICE
CON> EDIT_STRING $$$,$$$.$99.
DTR> PRINT NAME, REVIEW_CODE, CURRENT_SALARY,
CON> NEW_SALARY OF CURRENT

```

NAME	REVIEW CODE	CURRENT SALARY	NEW SALARY
HARRY	1	\$51,712.00	\$62,054.40
SAM	5	\$11,676.00	\$10,508.40
HORTENSE	4	\$18,497.00	\$18,497.00
SALLY	3	\$17,510.00	\$19,261.00
JANE	2	\$32,254.00	\$37,092.10

```
DTR>
```

In both the IF-THEN-ELSE and the CHOICE value expressions, the keyword THEN is optional but the ELSE clause is required.

18.1.1.7 FORMAT Value Expressions — These expressions consist of the keyword `FORMAT` followed by a value expression and then a `USING` clause with an edit string. The `USING` clause is optional. If you omit it, `DATATRIEVE` uses the default edit string for the field to carry out your operation.

A `FORMAT` value expression lets an edit string affect more than display operations. Depending on how you use it, a `FORMAT` edit string can tell `DATATRIEVE` how to store data and how to interpret data when searching for field values.

You can use the `FORMAT` value expression, for example, when you want `DATATRIEVE` to align numbers in a text field so that the least significant digit always occupies the rightmost character position in the field. Otherwise, `DATATRIEVE` fills a text field from left to right:

```
DTR> DECLARE INVOICE_NUMBER PIC X(6)
CON> VALID IF INVOICE_NUMBER NOT CONTAINING " ".
DTR> INVOICE_NUMBER = *."invoice number"
Enter invoice number: 456
Validation error for field INVOICE_NUMBER.
Re-enter invoice number: 000456
DTR> INVOICE_NUMBER = FORMAT *."invoice number" USING 9(6)
Enter invoice number: 456
DTR> PRINT INVOICE_NUMBER
```

```
INVOICE
NUMBER
```

```
000456
```

```
DTR>
```

The section later in this chapter on comparing and searching for data values contains an example of using a `FORMAT` value expression to search for date values corresponding to a day of the week.

18.1.2 Boolean Expressions

A Boolean expression specifies a relationship between value expressions. It consists of value expressions, relational operators, and Boolean operators:

- Value expressions specify the values you want to compare.
- Relational operators specify how you want to compare the values.
- Boolean operators join two or more Boolean expressions together or reverse the value of a Boolean expression.

All Boolean expressions contain at least two value expressions and at least one relational operator. Some contain Boolean operators. DATATRIEVE evaluates a Boolean expression as either true or false.

You use Boolean expressions in the following clauses, statements, and expressions:

- The WITH clause in a record selection expression or SELECT statement
- The IF clause of an IF-THEN-ELSE statement or value expression
- The CHOICE statement or value expression
- The COMPUTED BY or VALID IF clauses in a field definition
- The WHILE statement

18.1.2.1 Relational Operators — Relational operators let you compare value expressions, check whether a code value is in a table, check whether or not an RSE contains any records, or check whether or not a field value is missing.

Table 18-3 shows relational operators and describes their meaning in a Boolean expression. The characters a, b, and c in the table stand for value expressions used in the comparison.

Table 18-3: Relational Operators

Operator	Meaning	Example
a EQUAL b	True if a and b are equal	LAST_NAME EQUAL "SMITH"
a NOT_EQUAL b	True if a and b are not equal	STATE NOT_EQUAL "NH"
a GREATER_THAN b	True if a is greater than b	PRICE GREATER_THAN 50
a GREATER_EQUAL b	True if a is greater than or equal to b	PRICE GREATER_EQUAL 50
a LESS_THAN b	True if a is less than b	PRICE LESS_THAN 50000
a LESS_EQUAL b	True if a is less than or equal to b	PRICE LESS_EQUAL 50000

(continued on next page)

Table 18-3: Relational Operators (Cont.)

Operator	Meaning	Example
a BETWEEN b AND c	True if a is between b and c or if a is equal to b or c	PRICE BETWEEN 1000 AND 5000
a NOT BETWEEN b AND c	True if a is not between b and c or not equal to b or c	PRICE NOT BETWEEN 1000 AND 5000
a STARTING WITH b	True if the beginning characters of "a" match the characters specified by "b"	LAST_NAME STARTING WITH "S"
a CONTAINING b	True if a contains b	COMPANY CONTAINING "INC"
a NOT CONTAINING b	True if a does not contain b	COMPANY NOT CONTAINING "INC"
a BEFORE b	True if date value a is earlier than date value b	BIRTHDAY BEFORE "1-JAN-1910"
a AFTER b	True if date value a is later than date value b	BIRTHDAY AFTER "3/28/46"
a IN table-name	True if a is found in the specified table	DEPT_CODE IN DEPTS_TABLE
a NOT IN table-name	True if a is not found in the specified table	DEPT_CODE NOT IN DEPTS_TABLE
ANY rse	True if there are any records meeting RSE requirements	FAMILIES WITH ANY KIDS
NOT ANY rse	True if there are not any records meeting RSE requirements	FAMILIES WITH NOT ANY KIDS

(continued on next page)

Table 18-3: Relational Operators (Cont.)

Operator	Meaning	Example
a MISSING	True if a contains the value specified in the MISSING VALUE clause for a	SALARY_AMOUNT MISSING
a NOT MISSING	True if a does not contain the value specified in the MISSING VALUE clause for a	SALARY_AMOUNT NOT MISSING

To save typing time, you can use the following abbreviations and symbols for relational operators:

EQUAL —————> EQ or =
 NOT_EQUAL —————> NE or NOTEQUAL
 GREATER_THAN —————> GT or >
 GREATER_EQUAL —————> GE
 LESS_THAN —————> LT or <
 LESS_EQUAL —————> LE
 BETWEEN —————> BT
 CONTAINING —————> CONT

When you use relational operators that compare for lesser or greater values in character strings, each character has a value. Appendix C lists all the characters that can appear in character strings according to their relative “values.”

If you refer to Appendix C, you see that lowercase letters are considered “greater than” uppercase letters. All relational operators except for CONTAINING, NOT CONTAINING, BEFORE, and AFTER are case-sensitive. EQUAL “SMITH” and EQUAL “Smith” are not equivalent expressions, but CONTAINING “SMITH” and CONTAINING “Smith” are equivalent. The date values compared with the BEFORE and AFTER operators are displayed like text strings but they are not stored as text. You do not have to pay attention to character values when you enter date value expressions.

The ANY and NOT ANY relational operators are particularly useful when a record search depends on values contained in variable length lists (fields defined with both the OCCURS and DEPENDING ON clauses). In the following example, the field KIDS is a variable length list in the FAMILIES domain:

```
DTR> PRINT FAMILIES WITH ANY KIDS WITH
CON> KID_NAME CONT "RALPH"
```

FATHER	MOTHER	NUMBER KIDS	KID NAME	AGE
JIM	ANN	2	URSULA	7
			RALPH	3

18.1.2.2 Boolean Operators — You use the AND and OR Boolean operators to join Boolean expressions and the NOT Boolean operator to reverse the value of a Boolean expression.

When you join Boolean expressions with AND, the resulting expression is true only if *both* expressions are true:

```
DTR> PRINT LAST_NAME, TOWN, STATE OF FIRST 5 ADDRESS_LIST WITH
CON> TOWN = "Milton" AND STATE = "NH"
```

LAST NAME	TOWN	STATE
Reynolds	Milton	NH
Frydman	Milton	NH
Gehr	Milton	NH
Moen	Milton	NH
Brown	Milton	NH

```
DTR>
```

When you join Boolean expressions with OR, the resulting expression is true if *either one* of the expressions is true:

```
DTR> PRINT LAST_NAME, TOWN, STATE OF FIRST 5 ADDRESS_LIST WITH
CON> TOWN = "Milton" OR STATE = "NH"
```

LAST NAME	TOWN	STATE
Toliver	Chocorua	NH
Smith	Chocorua	NH
Dietrich	Boscawen	NH
Kilpatrick	Marlow	NH
Nash	Meadows	NH

When you precede a Boolean expression with NOT, the resulting expression is true if the expression following NOT is false. In the following example, CURRENT_SALARIES is a view domain based on SALARY_HISTORY that accesses salary information for current employees:

```
DTR> PRINT EMPLOYEE_ID, SALARY_AMOUNT OF  
CON> CURRENT_SALARIES WITH  
CON> SALARY_AMOUNT NOT LT 85000
```

EMPLOYEE ID	SALARY AMOUNT
00228	\$85,150.00
00415	\$86,124.00
00204	\$87,143.00
00359	\$93,340.00

```
DTR>
```

You can enclose parts of compound Boolean expressions in parentheses. DATATRIEVE evaluates Boolean expressions in parentheses before evaluating other Boolean expressions. If a Boolean expression contains Boolean operators as well as parentheses, DATATRIEVE evaluates the compound expression in this order:

1. Expressions enclosed in parentheses
2. Expressions preceded by NOT
3. Expressions combined with AND
4. Expressions combined with OR

Use parentheses to group Boolean expressions into compound Boolean expressions to ensure that your expressions identify the records you want. For example:

```
DTR> FIND ADDRESS_LIST WITH TOWN = "Milton" AND  
CON> (STATE = "NH" OR STATE = "MA")  
[20 records found]  
DTR> PRINT ALL LAST_NAME, TOWN, STATE
```

(continued on next page)

LAST NAME	TOWN	STATE
Reynolds	Milton	NH
Frydman	Milton	NH
Gehr	Milton	NH
.	.	.
.	.	.
Klein	Milton	MA
.	.	.
.	.	.
.	.	.

DTR>

Here is what happens when the parentheses are not included. The true condition for the Boolean expression is met if a record includes either Milton, NH in the address, or only MA as the state:

```
DTR> FIND ADDRESS_LIST WITH TOWN = "Milton" AND STATE = "NH" OR
CON> STATE = "MA"
[49 records found]
DTR> PRINT ALL LAST_NAME, TOWN, STATE
```

LAST NAME	TOWN	STATE
Myotte	Bennington	MA
Siciliano	Farmington	MA
Reynolds	Milton	NH
.	.	.
.	.	.
.	.	.

DTR>

18.2 Computing Sums and Other Statistics

This section shows you how to use statistical value expressions and the SUM statement.

18.2.1 Statistical Value Expressions

DATATRIEVE statistical functions let you summarize and calculate statistical information from fields in your records. Usually, you enter a statistical function along with the name of a record field or a variable (TOTAL SALARY_AMOUNT, for example) and DATATRIEVE calculates the result. The exceptions are the functions COUNT and RUNNING COUNT, which you use by themselves. Table 18-4 lists DATATRIEVE statistical functions and the values they compute.

Table 18-4: Statistical Functions and Results

Function	Result
AVERAGE	Averages the values in the expression
MAX	Identifies the largest value in the expression
MIN	Identifies the smallest value in the expression
STD_DEV	Computes the standard deviation for the values in the expression
TOTAL	Totals the values in the expression
RUNNING TOTAL	In a PRINT statement loop, displays the running total of the values in the expression
COUNT	Counts the number of records in a record source
RUNNING COUNT	In a PRINT statement loop, displays a running count for the items displayed

When you use statistical value expressions, you must follow each one with the keyword **OF** and then an **RSE** (**TOTAL SALARY_AMOUNT OF CURRENT_SALARIES**, for example). The exception to this rule is when the value expressions apply to records in the **CURRENT** collection.

Example 18-1 shows you how to use statistical value expressions to obtain information. The example uses data from two view domains, **CURRENT_JOBS** and **CURRENT_SALARIES**.

CURRENT_SALARIES contains one record in which **SALARY_AMOUNT** is equal to 0. Because the field definition for **SALARY_AMOUNT** (in **JOB_HISTORY_REC**) specifies **MISSING VALUE = 0**, **DATATRIEVE** does not include that record in statistical computations based on **SALARY_AMOUNT**. When **DATATRIEVE** displays a statistical result for **SALARY_AMOUNT**, it precedes that result with a message telling you that it is not based on values from all the records.

Example 18-1: Using Statistical Value Expressions

```
DTR> SHOW CURRENT_JOBS
DOMAIN CURRENT_JOBS OF JOB_HISTORY
01 CURRENT_JOB OCCURS FOR JOB_HISTORY WITH
    JOB_END MISSING SORTED BY DEPARTMENT_CODE, JOB_CODE.
    03 DEPARTMENT_CODE FROM JOB_HISTORY.
    03 JOB_CODE FROM JOB_HISTORY.
    03 EMPLOYEE_ID FROM JOB_HISTORY.
    03 JOB_START FROM JOB_HISTORY.
    03 SUPERVISOR_ID FROM JOB_HISTORY.
```

```
DTR> SHOW CURRENT_SALARIES
DOMAIN CURRENT_SALARIES OF SALARY_HISTORY
01 CURRENT_SALARY OCCURS FOR SALARY_HISTORY WITH
    SALARY_END MISSING.
    03 EMPLOYEE_ID FROM SALARY_HISTORY.
    03 REVIEW_CODE FROM SALARY_HISTORY.
    03 SALARY_AMOUNT FROM SALARY_HISTORY.
```

```
DTR> READY CURRENT_JOBS, CURRENT_SALARIES
DTR> PRINT COUNT OF CURRENT_JOBS
```

COUNT

337

```
DTR> PRINT COUNT OF CURRENT_JOBS, COUNT OF CURRENT_SALARIES
```

COUNT COUNT

337 337

```
DTR> PRINT COUNT OF CURRENT_JOBS ("Job"/"Count"),
CON> COUNT OF CURRENT_SALARIES ("Salary"/"Count")
```

Job Salary
Count Count

337 337

```
DTR> PRINT TOTAL SALARY_AMOUNT OF
CON> CURRENT_SALARIES USING $$$$, $$$, $$$ .99
```

TOTAL
SALARY
AMOUNT

```
[Function computed using 336 of 337 values.]
$8,967,907.00
```

DTR) PRINT AVERAGE SALARY_AMOUNT OF CURRENT_SALARIES

AVERAGE
SALARY
AMOUNT

[Function computed using 336 of 337 values.]
\$26,611.00

DTR) PRINT MAX SALARY_AMOUNT OF CURRENT_SALARIES

MAX
SALARY
AMOUNT

[Function computed using 336 of 337 values.]
\$93,340.00

DTR) PRINT MIN SALARY_AMOUNT OF CURRENT_SALARIES

MIN
SALARY
AMOUNT

[Function computed using 336 of 337 values.]
\$7,205.00

DTR) PRINT STD_DEV SALARY_AMOUNT OF CURRENT_SALARIES

STANDARD
DEVIATION
SALARY
AMOUNT

[Function computed using 336 of 337 values.]
1.7113E+04

DTR) PRINT STD_DEV SALARY_AMOUNT OF
CON) CURRENT_SALARIES USING \$\$\$\$,\$\$\$.99

STANDARD
DEVIATION
SALARY
AMOUNT

[Function computed using 336 of 337 values.]
\$17,113.30

The next section explains an option, the SUM statement, you can use only with a CURRENT collection. The section also contains some examples of statistical value expressions that use fields from a CURRENT collection.

18.2.2 Using the SUM Statement and Statistical Value Expressions with the CURRENT Collection

When you are working with a CURRENT collection, you can use the SUM statement to sort and total data. Example 18-2 shows you how to enter a SUM statement and statistical value expressions using a CURRENT collection. The records in the collection join fields from the views CURRENT_JOBS and CURRENT_SALARIES. The comment lines in the example begin with an exclamation point (!) and provide information about the input that follows.

Example 18-2: Using the SUM Statement and Statistical Value Expressions with the CURRENT Collection

```
DTR> ! Create a CURRENT collection and print its records.
DTR> !
DTR> FIND CURRENT_JOBS CROSS CURRENT_SALARIES OVER
CON> EMPLOYEE_ID WITH JOB_CODE = "CLRK" REDUCED TO
CON> DEPARTMENT_CODE, EMPLOYEE_ID, SALARY_AMOUNT
[19 records found]
DTR> PRINT ALL
```

DEPARTMENT CODE	EMPLOYEE ID	SALARY AMOUNT
ELMC	00181	\$19,953.00
ELMC	00333	\$15,968.00
MBMS	00319	\$12,350.00
MCBM	00396	\$13,561.00
MCBM	00483	\$14,745.00
MSCI	00263	\$19,057.00
MSCI	00337	\$16,987.00
MSCI	00392	\$13,162.00
SUNE	00391	\$14,173.00
SUSA	00279	\$15,571.00
SUSA	00364	\$15,315.00
SUSA	00382	\$15,392.00
SUSA	00404	\$12,135.00
SUSA	00406	\$14,306.00
SUSO	00187	\$13,067.00
SUSO	00193	\$17,960.00
SUSO	00245	\$14,794.00
SUSO	00268	\$18,933.00
SUSO	00289	\$15,560.00

```
DTR> !
DTR> ! Begin a SUM statement with the keyword SUM and
DTR> ! enter the item (or items) to be totaled. Then enter
DTR> ! the keyword BY followed by the item (or items) on
DTR> ! which you want the information sorted.
DTR> !
DTR> SUM SALARY_AMOUNT BY DEPARTMENT_CODE
```

DEPARTMENT CODE	TOTAL SALARY AMOUNT
ELMC	35921
MBMS	12350
MCBM	28306
MSCI	49206
SUNE	14173
SUSA	72719
SUSO	80314
	292989

DTR) !
DTR) ! You can include an edit string for each of the items
DTR) ! being totaled.
DTR) !
DTR) SUM SALARY_AMOUNT USING \$\$\$,\$\$.99 BY DEPARTMENT_CODE

DEPARTMENT CODE	TOTAL SALARY AMOUNT
ELMC	\$35,921.00
MBMS	\$12,350.00
MCBM	\$28,306.00
MSCI	\$49,206.00
SUNE	\$14,173.00
SUSA	\$72,719.00
SUSO	\$80,314.00
	\$292,989.00

DTR) !
DTR) ! You can also specify the column headers you want
DTR) ! for each field being totaled.
DTR) !
DTR) SUM SALARY_AMOUNT ("PAYROLL FOR"/"CLERKS") USING
CON) \$\$\$,\$\$.99 BY DEPARTMENT_CODE

DEPARTMENT CODE	PAYROLL FOR CLERKS
ELMC	\$35,921.00
MBMS	\$12,350.00
MCBM	\$28,306.00
MSCI	\$49,206.00
SUNE	\$14,173.00
SUSA	\$72,719.00
SUSO	\$80,314.00
	\$292,989.00

(continued on next page)

```

DTR> !
DTR> ! You can include the keyword 1 as one of the items
DTR> ! listed following SUM. When you do this, DATATRIEVE
DTR> ! displays a count to indicate how many records were
DTR> ! used to compute the total for each sort key.
DTR> !
DTR> SUM 1 ("TOTAL"/"CLERKS"),
CON>     SALARY_AMOUNT ("PAYROLL FOR"/"CLERKS") USING
CON>     $$$,$$$.$99 BY DEPARTMENT_CODE

```

DEPARTMENT CODE	TOTAL CLERKS	PAYROLL FOR CLERKS	TOTAL CLERKS
ELMC	2	\$35,921.00	
MBMS	1	\$12,350.00	
MCBM	2	\$28,306.00	
MSCI	3	\$49,206.00	
SUNE	1	\$14,173.00	
SUSA	5	\$72,719.00	
SUSO	5	\$80,314.00	
			19
		\$292,989.00	

```

DTR> !
DTR> ! When you use statistical expressions with the CURRENT
DTR> ! collection, you do not need to specify an OF rse clause
DTR> ! for each expression.
DTR> !
DTR> PRINT AVERAGE SALARY_AMOUNT

```

```

AVERAGE
SALARY
AMOUNT

$15,420.47

```

```

DTR> PRINT STD_DEV SALARY_AMOUNT USING $$$,$$$.$99

```

```

STANDARD
DEVIATION
SALARY
AMOUNT

$2,277.05

```

```

DTR>

```

18.3 Storing and Displaying Date and Time

This section discusses the options you have when storing or searching for values in fields defined as USAGE DATE. It also shows you various ways you can display values from those fields. The information provided here supplements the explanation of date fields in Chapter 9. You might want to review that explanation before reading the following sections.

18.3.1 Storing and Displaying Values in Date Fields

There are several ways to store a date. You can:

- Enter the date as a literal

```
DATE_FIELD = "28-JAN-1952"
```

- Enter the date as a date value expression (TODAY, TOMORROW, YESTERDAY)

```
DATE_FIELD = "TODAY"
```

To store the time, in addition to the date, you can:

- Enter the current date and time with the value expression NOW

```
DATE_FIELD = "NOW"
```

- Enter any date and time using the function FN\$DATE and a character string

The character string *must* adhere to the format DD-MMM-YYYY hh:mm:ss.cc. In this format, DD is the day, MMM (each letter capitalized) is the month, YYYY is the year, hh is the hour (00 for 12 o'clock a.m. to 23 for 11 o'clock p.m.), mm is the minute, ss is the second, and cc is the hundredth of a second.

```
DATE_FIELD = FN$DATE("12-JAN-1962 13:30:00.00")
```

If you store time in a date field, you can display it using the edit string X(23). If you do not supply an edit string for a date field either in the field definition or in a PRINT statement, DATATRIEVE uses the edit string DD-MMM-YYYY.

Note

You can assign a value to the logical name DTR\$DATE_INPUT to control the way DATATRIEVE converts date values represented by a string of numbers. By default, for example, DATATRIEVE interprets the value "03/12/09" as March 12, 1909. If the following assignment is in effect, DATATRIEVE interprets the values "03/12/09" or "031209" as December 3, 1909:

```
ASSIGN DMY DTR$DATE_INPUT
```

For more information on using the logical name DTR\$DATE_INPUT, refer to the EDIT_STRING article in the *VAX DATATRIEVE Reference Manual*.

Example 18-3 illustrates the options you have when displaying values in date fields.

Example 18-3: Storing and Displaying Values in Fields Defined as USAGE DATE

```
DTR> DECLARE DATE_FIELD USAGE DATE.  
DTR> DATE_FIELD = "3/28/47"  
DTR> PRINT DATE_FIELD
```

```
DATE  
FIELD
```

```
28-Mar-1947
```

```
DTR> PRINT DATE_FIELD USING NN/DD/YY
```

```
DATE  
FIELD
```

```
3/28/47
```

```
DTR> PRINT DATE_FIELD USING MMMD,BYYYY
```

```
DATE  
FIELD
```

```
Mar. 28, 1947
```

```
DTR> PRINT DATE_FIELD USING M(9)BDD,BYYYY
```

```
DATE  
FIELD
```

```
March 28, 1947
```

```
DTR> PRINT DATE_FIELD USING YY
```

```
DATE  
FIELD
```

```
47
```

```
DTR> PRINT DATE_FIELD USING MMM/YYYY
```

```
DATE  
FIELD
```

```
Mar/1947
```

DTR> PRINT DATE_FIELD USING WWW

DATE
FIELD

Fri

DTR> PRINT DATE_FIELD USING WWW/MMMBDD

DATE
FIELD

Fri/Mar 28

DTR> PRINT DATE_FIELD USING JJJ

DATE
FIELD

87

DTR> DATE_FIELD = "TODAY"

DTR> PRINT DATE_FIELD

DATE
FIELD

6-Apr-1984

DTR> DATE_FIELD = "YESTERDAY"

DTR> PRINT DATE_FIELD

DATE
FIELD

5-Apr-1984

DTR> DATE_FIELD = "TOMORROW"

DTR> PRINT DATE_FIELD

DATE
FIELD

7-Apr-1984

DTR> DATE_FIELD = "NOW"

DTR> PRINT DATE_FIELD USING X(23)

DATE
FIELD

6-Apr-1984 11:57:48.56

DTR> DATE_FIELD = FN\$DATE("28-MAR-1947 04:59:06.01")

DTR> PRINT DATE_FIELD USING X(23)

DATE
FIELD

28-Mar-1947 04:59:06.01

DTR>

18.3.2 Comparing and Searching for Date Values

You can compare date values using all the relational operators except IN, NOT IN, ANY, and NOT ANY. BEFORE and AFTER are two operators especially designed for dates. You can also get useful information by subtracting dates, with the statistical functions MIN and MAX, and with the FORMAT value expression. Example 18-4 illustrates some techniques you can use with date values. The comment lines in the example begin with an exclamation point (+) and provide information about the input that follows.

Example 18-4: Comparing and Searching for Date Values

```
DTR> READY CURRENT_JOBS
DTR> !
DTR> ! Print the records for current employees who started
DTR> ! work before and after a specified date.
DTR> !
DTR> PRINT CURRENT_JOBS WITH JOB_START BEFORE "12/31/82"
```

DEPARTMENT CODE	JOB CODE	EMPLOYEE ID	JOB START	SUPERVISOR ID
ADMN	DSUP	00472	27-Apr-1981	00225
ADMN	EENG	00300	11-Feb-1982	00225
ADMN	EENG	00188	8-Apr-1982	00225
.
SUWE	SPGM	00284	16-Dec-1980	00230

```
DTR> PRINT CURRENT_JOBS WITH JOB_START AFTER "12/31/82"
```

DEPARTMENT CODE	JOB CODE	EMPLOYEE ID	JOB START	SUPERVISOR ID
ADMN	PRSD	00225	3-Jan-1983	00225
PERL	DMGR	00241	3-Jan-1983	00415
PHRN	EENG	00437	4-Feb-1983	00201

```
DTR> !
DTR> ! Find out how long current employees have been working
DTR> ! for the company. (WHO_IS_IT is a domain table that
DTR> ! accesses the employee name from the EMPLOYEES domain.)
DTR> !
DTR> DECLARE DAYS_WORKING_HERE COMPUTED BY
CON> ("TODAY" - JOB_START).
DTR> FOR CURRENT_JOBS SORTED BY JOB_START
CON> PRINT EMPLOYEE_ID VIA WHO_IS_IT, DAYS_WORKING_HERE
```

EMPLOYEE NAME			DAYS WORKING HERE
Ackerman	Ellen	C	1923
Kinmonth	Louis		1908
Roberts	Norman	U	1873
.	.	.	.
.	.	.	.
Stornelli	Franklin		455

DTR> !
DTR> ! A field computed by dates has a default edit string
DTR> ! much longer than what you need for a display. Reenter
DTR> ! the statement with an edit string for the field
DTR> ! DAYS_WORKING_HERE.
DTR> !
DTR> FOR CURRENT_JOBS SORTED BY JOB_START
CON> PRINT EMPLOYEE_ID VIA WHO_IS_IT,
CON> DAYS_WORKING_HERE USING ZZ,ZZ9

EMPLOYEE NAME			DAYS WORKING HERE
Ackerman	Ellen	C	1,923
Kinmonth	Louis		1,908
Roberts	Norman	U	1,873
.	.	.	.
.	.	.	.
Stornelli	Franklin		455

DTR> !
DTR> ! Find out which current employee has been working
DTR> ! for the company the longest period of time.
DTR> !
DTR> PRINT CURRENT_JOBS WITH JOB_START = MIN JOB_START OF
CON> CURRENT_JOBS

DEPARTMENT CODE	JOB CODE	EMPLOYEE ID	JOB START	SUPERVISOR ID
SUWE	JNTR	00318	28-Jan-1979	00230

DTR> !
DTR> ! Find out which employee was hired most recently.
DTR> !
DTR> PRINT CURRENT_JOBS WITH JOB_START = MAX JOB_START OF
CON> CURRENT_JOBS

DEPARTMENT CODE	JOB CODE	EMPLOYEE ID	JOB START	SUPERVISOR ID
PHRN	EENG	00437	4-Feb-1983	00201

(continued on next page)

```

DTR> !
DTR> ! Print records for employees hired in 1982.
DTR> !
DTR> PRINT CURRENT_JOBS WITH JOB_START CONT "1982"

```

DEPARTMENT CODE	JOB CODE	EMPLOYEE ID	JOB START	SUPERVISOR ID
ADMN	EENG	00188	8-Apr-1982	00225
ADMN	EENG	00300	11-Feb-1982	00225
ADMN	MENG	00190	25-Feb-1982	00225
ADMN	SPGM	00488	6-Mar-1982	00225
ADMN	VPSD	00267	28-Feb-1982	00225
ADMN	VPSD	00439	26-Nov-1982	00225
ADMN	VPSD	00204	24-Jan-1982	00225
ELEL	APGM	00377	26-Jan-1982	
ELEL	EENG	00428	10-Jan-1982	00356
ELEL	EENG	00238	2-Feb-1982	00356
.
.
.
SUWE	EENG	00293	20-Sep-1982	00230

```

DTR> !
DTR> ! Form a collection of records for employees starting
DTR> ! work on a specific day of the week. Print their
DTR> ! employee ID numbers and start dates.
DTR> !
DTR> FIND CURRENT_JOBS WITH FORMAT JOB_START USING WWW CONT "MON"
[45 records found]
DTR> PRINT ALL EMPLOYEE_ID, JOB_START USING DD-MMM-YYYY/WWW

```

EMPLOYEE ID	JOB START
00472	27-Apr-1981/Mon
00225	3-Jan-1983/Mon
00435	17-Nov-1980/Mon
00211	25-Jan-1982/Mon
.	.
.	.
.	.
00293	20-Sep-1982/Mon

```
DTR>
```

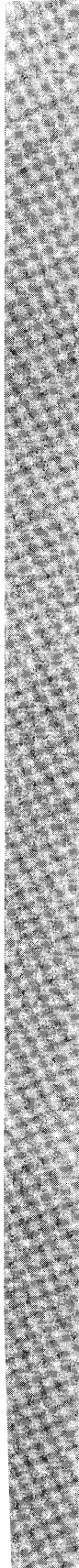
18.3.3 Subtracting Values from a Date Field

When you subtract a value from a date, DATATRIEVE converts the value to a date and then subtracts it. If you want to subtract days from dates, you must add the number of days as a negative number:

```
DTR> BALANCE_DATE = BALANCE_DATE + (-1)
```

Part VI
Formatting Displays and Writing Reports





Improving Screen Displays and Controlling Output **19**

This chapter discusses how you can change DATATRIEVE display defaults to suit your needs. It also summarizes information about PRINT statement variations and options. You can use the tables in this chapter as references when you are creating your own displays and reports. This chapter does not cover defining a form to use with DATATRIEVE. If you are interested in displaying data on a VAX TDMS or VAX FMS form, refer to the chapter on using forms in the *VAX DATATRIEVE User's Guide*.

19.1 Optimizing Space in Display Lines

When you display records, you might find that the fields from each record do not all fit on one display line on your terminal screen. The result is that some fields wrap to the next line and that headers for some fields either do not appear at all or are inserted in available space between other headers. A display like this is not very attractive and often difficult to read.

If you are interested in seeing only a subset of the fields in a record, you can try listing the names of the fields you want to see following the keyword PRINT. If these fit on one line, your problem is solved. If they do not or if you want to see the entire record, See the following sections which discuss some other things you can try.

19.1.1 Adjusting Screen Width and the Columns-Page Setting

If your terminal screen width is set to 80 characters (the default most people have), you can increase this to 132 characters. The extra 52 columns might be enough to accommodate what you want to look at. Changing screen width is a 2-step process. You have to tell the operating system to adjust your screen display, and you have to tell DATATRIEVE to space its output across the specified number of columns. The order in which you do the steps does not matter:

```
DTR> FN$WIDTH (132)
DTR> SET COLUMNS_PAGE=132
```

The reduced character size that comes with the 132-character setting is not to everyone's liking. In addition, some record displays require more than 132 columns. If you want to set your screen width back to 80 columns, use the FN\$WIDTH function and the SET COLUMNS_PAGE command again, but specify 80 in place of 132.

19.1.2 Using the LIST Statement

The LIST statement displays each field from the record on a separate line. Rather than displaying a column header for each field, it prints the field name followed by a colon (:) and then the field contents. When the elementary field names are descriptive of the data in the field, the LIST statement can improve the readability of long record displays:

```
DTR> FIND EMPLOYEES WITH EMPLOYEE_ID = "00181"
[1 record found]
DTR> SELECT
DTR> PRINT
```

SOCIAL			ADDRESS			
ID	LAST NAME	FIRST NAME INIT	DATA	ZIP	SEX	
SECURITY						
00181	Reynolds	Louis E	Apartment 78C			
	63 Derry Rd.	Milton	NH	03851	M	393 98 1984
	12/11/52					

DTR> LIST

```
EMPLOYEE_ID      : 00181
LAST_NAME        : Reynolds
FIRST_NAME       : Louis
MIDDLE_INITIAL   : E
ADDRESS_DATA     : Apartment 78C
STREET           : 63 Derry Rd.
TOWN             : Milton
STATE            : NH
ZIP              : 03851
SEX              : M
SOCIAL_SECURITY  : 393 98 1984
BIRTHDAY         : 12/11/52
```

DTR>

19.1.3 Writing a Simple Procedure to Segment Record Display

If you plan to print an entire record from a domain or view frequently, you can write a short procedure to display the record neatly on more than one line. This method lets you use a PRINT statement, which accesses any column headers defined in the record definition, or lets you override these with ones of your own choosing. In the following example, the procedure prints a string literal of 80 hyphens to set off each line. The last hyphen in the first line of the literal is a continuation character, which tells DATATRIEVE that the literal is continued on the next line:

```
DTR> SHOW PRINT_EMP
PROCEDURE PRINT_EMP
BEGIN
  PRINT "-----"
  PRINT EMPLOYEE_ID, LAST_NAME, FIRST_NAME, MIDDLE_INITIAL
  PRINT "-----"
  PRINT ADDRESS_DATA, STREET, TOWN, STATE, ZIP
  PRINT "-----"
  PRINT SEX, SOCIAL_SECURITY, BIRTHDAY
  PRINT "-----"
END
END_PROCEDURE
```

(continued on next page)

```
DTR> FIND EMPLOYEES WITH EMPLOYEE_ID = "00181"
[1 record found]
DTR> SELECT
DTR> PRINT
```

ID	LAST NAME	FIRST NAME INIT	ADDRESS DATA	ZIP	SEX	SOCIAL SECURITY
00181	Reynolds	Louis E	Apartment 78C 63 Derry Rd. 12/11/52	NH 03851	M	393 98 1984

```
DTR> :PRINT_EMP
```

ID	LAST NAME	FIRST NAME INIT
00181	Reynolds	Louis E

ADDRESS DATA	STREET	TOWN	STATE	ZIP
Apartment 78C	63 Derry Rd.	Hudson	NH	03851

SEX	SOCIAL SECURITY	BIRTHDAY
M	393 98 1984	12/11/52

```
DTR>
```

This method is effective when you are displaying one record at a time. It is not very helpful when you want to print more than one record with a single statement. In this case, DATATRIEVE only prints the column headers for the first record displayed.

19.1.4 Overriding Column Header Defaults with the PRINT Statement

When you enter a PRINT statement that simply lists the items you want to display or one that prints an entire record, DATATRIEVE checks each field definition for a QUERY_HEADER clause. If the field definition contains this clause, DATATRIEVE displays the specified column header above the field value. If the field definition does not contain a QUERY_HEADER clause, DATATRIEVE makes a column header out of the field name.

If the column header DATATRIEVE uses is longer than the largest value that can appear in a field, you can conserve some space in your display line by doing one of three things:

- Edit the record definition to include a QUERY_HEADER clause that specifies a header no longer than the length of the field.

Chapter 9 tells you how to edit a record definition and write QUERY_HEADER clauses.

- Specify another header for the field in your PRINT statement.

If the field REVIEW_CODE were a 1-character field, for example, you could enter PRINT REVIEW_CODE ("C") or PRINT REVIEW_CODE ("R"/"C") to make sure the header for the field is also one character in length.

- Specify in your PRINT statement that the field values are displayed without a header.

You do this by typing the field name followed by a hyphen enclosed in parentheses, (-); for example, PRINT REVIEW_CODE (-).

The following example defines and displays three variable fields to illustrate these options. You can manipulate the display of record fields in the same way:

```
DTR> DECLARE WIDGET_CODE PIC X.
DTR> DECLARE WIDGET_NAME PIC X(5).
DTR> DECLARE WIDGET_PRICE PIC 99V99
CON>          EDIT_STRING $$$ .99.
DTR> WIDGET_CODE = "Z"
DTR> WIDGET_NAME = "WHIZZ"
DTR> WIDGET_PRICE = 10.50
DTR> PRINT WIDGET_CODE, WIDGET_NAME, WIDGET_PRICE

WIDGET WIDGET WIDGET
CODE   NAME  PRICE

   Z   WHIZZ  $10.50

DTR> PRINT WIDGET_CODE (-), WIDGET_NAME (-), WIDGET_PRICE (-)
Z WHIZZ $10.50

DTR> PRINT WIDGET_CODE ("C"/"O"/"D"/"E"),
CON>          WIDGET_NAME ("NAME"),
CON>          WIDGET_PRICE ("PRICE")

C
O
D
E NAME  PRICE

Z WHIZZ $10.50

DTR>
```

19.1.5 Using Edit Strings to Optimize Display Space

When you enter a PRINT statement that simply lists the items you want to display or one that prints an entire record, DATATRIEVE checks each field definition for an EDIT_STRING clause. If the field definition contains this clause, DATATRIEVE displays the field contents according to the specified edit string. If the field definition does not contain this clause, DATATRIEVE formats field contents according to the PIC or USAGE clauses.

For some fields defined with the COMPUTED BY clause and some very long numeric fields, you might find that the values stored in the field are much smaller than the space allotted to them in the display line. This situation can also arise when you are printing value expressions calculated from two or more field values. In these cases, you can save some space by including an edit string in your PRINT statement. Take care that you do not inadvertently truncate some values by doing this, however. The following example defines and displays variable fields to illustrate this technique, which works just as well when you are displaying record fields:

```
DTR> DECLARE DATE_VARIABLE1 USAGE DATE.
DTR> DATE_VARIABLE1 = "TODAY"
DTR> !
DTR> DECLARE DATE_VARIABLE2 USAGE DATE.
DTR> DATE_VARIABLE2 = "23-JUL-1984"
DTR> !
DTR> DECLARE BIG_NUMBER USAGE QUAD.
DTR> BIG_NUMBER = 1234567890
DTR> !
DTR> PRINT DATE_VARIABLE2 - DATE_VARIABLE1, BIG_NUMBER
```

```

                BIG
                NUMBER
106                1234567890
```

```
DTR> !
DTR> ! Edit strings that are too small can cause problems.
DTR> !
DTR> PRINT DATE_VARIABLE2 - DATE_VARIABLE1 USING 9(6),
CON>    BIG_NUMBER USING ZZZ,ZZZ,ZZ9
```

```

                BIG
                NUMBER
```

```
000106 *****
```

```
DTR> !
DTR> ! Edit strings adapted to the size of the values being
DTR> ! displayed can optimize display space.
DTR> !
DTR> PRINT DATE_VARIABLE2 - DATE_VARIABLE1 USING Z(5)9,
CON>    BIG_NUMBER USING 9(10)
```

```
BIG  
NUMBER
```

```
106 1234567890
```

```
DTR> !  
DTR> ! The next PRINT statement takes two characters from  
DTR> ! the first number and adds some editing characters to the  
DTR> ! second number.  
DTR> !  
DTR> PRINT DATE_VARIABLE2 - DATE_VARIABLE1 USING Z(3)9,  
CON> BIG_NUMBER USING ZZZ,ZZZ,ZZZ,ZZ9
```

```
BIG  
NUMBER
```

```
106 1,234,567,890
```

```
DTR>
```

19.1.6 Using Concatenation Characters to Conserve Line Space

Sometimes you can conserve space in a display line by using the **concatenation characters** (|, | |, and | | |) to join fields and literals into a continuous text string. The difference between the three concatenation characters is the way they treat trailing spaces in the value that precedes them and whether they add any spaces between values they join:

- A single bar (|) does nothing to the values except join them.
- A double bar (| |) suppresses any trailing spaces in the value that precedes it and does nothing to the value that follows it.
- A triple bar (| | |) suppresses any trailing spaces in the value that precedes it, inserts one space, and does nothing to the value that follows it.

When you join fields and literals this way, you form a **concatenation value expression**. As is the case with any value expression that is neither a field name nor a statistical value expression based on one field name, DATATRIEVE does not supply a default header for the result. If you want one, you must supply a column header in your PRINT statement. In the following example, a procedure using concatenation value expressions logically groups fields from each record in EMPLOYEES so that groups of records display in more readable form:

```
DTR> SHOW CONCATENATE
PROCEDURE CONCATENATE
  BEGIN
  PRINT SKIP
  PRINT EMPLOYEE_ID!!!FIRST_NAME!!!MIDDLE_INITIAL!!!LAST_NAME, SKIP
  PRINT ADDRESS_DATA!!! " !STREET!!!", " !TOWN!!!", " !STATE!!!ZIP, SKIP
  PRINT SEX (-), SOCIAL_SECURITY (-) USING XXX-XX-XXXX,
    BIRTHDAY (-)
  END
END_PROCEDURE
```

```
DTR> FOR FIRST 3 EMPLOYEES :CONCATENATE
```

```
00164 Alvin A Toliver
```

```
  146 Parnell Place, Chocorua, NH 03817
```

```
M 763-08-0064  3/28/47
```

```
00165 Terry D Smith
```

```
  120 Tenby Dr., Chocorua, NH 03817
```

```
M 179-97-8016  5/15/54
```

```
00166 Rick Dietrich
```

```
  19 Union Square, Boscawen, NH 03301
```

```
M 902-87-8080  3/20/54
```

```
DTR>
```

You can also use concatenation characters to create text literals that are longer than 253 characters. Because you are allowed to enter only up to 255 characters in a DATATRIEVE command or statement, most of the values you are joining should be entered as field names rather than text literals. When you display the combined values, you must tell DATATRIEVE how many characters of the string you want on each display line. Use the T edit-string character followed by a repeat count to do this:

```
DTR> DECLARE A PIC X(80).
DTR> DECLARE B PIC X(80).
DTR> DECLARE C PIC X(80).
DTR> DECLARE D PIC X(80).
DTR> !
DTR> PRINT *.A!!!*.B!!!*.C!!!*.D USING T(40)
Enter A: When DATATRIEVE joins the values in A, B, C, and D, It
Enter B: suppresses any trailing spaces in A, B, and C and
Enter C: inserts one space. It displays their combined values using
Enter D: up to 40 characters per line and without breaking words.
When DATATRIEVE joins the values in A,
B, C, and D, it suppresses any trailing
spaces in A, B, and C and inserts one
space. It displays their combined values
using up to 40 characters per line and
without breaking words.

DTR>
```

19.2 PRINT Statement Options

This section summarizes information about the PRINT statement.

19.2.1 PRINT Statement Format and Print List Elements

When you specify fields or other items you want to print, you are creating a **print list**. If the print list contains more than one item, you must separate the items with commas.

The format you use for a PRINT statement depends on the source of the values you want to display and whether or not your statement includes a print list:

- If the source of your values is a selected record from the CURRENT collection:
 - To display an entire record, type PRINT:

```
FIND an-rse
SELECT a-record
PRINT
```


- To display a subset of fields from the record, type PRINT followed by a print list:

```
FIND an-rse
SELECT a-record
PRINT print-list
```

- If the source of your values is the CURRENT collection:

- To display entire records, type PRINT ALL:

```
FIND an-rse
PRINT ALL
```

- To specify a subset of fields from each record, type PRINT ALL followed by a print list:

```
FIND an-rse
PRINT ALL print-list
```

- If the source of your values is a FOR statement RSE:

- To display entire records, type PRINT:

```
FOR an-rse
PRINT
```

- To specify a subset of fields from each record, type PRINT followed by a print list:

```
FOR an-rse
PRINT print-list
```

- If the source of your values is an RSE in the PRINT statement itself:

- To display entire records, type PRINT followed by the RSE:

```
PRINT an-rse
```

- To specify a subset of fields from each record, type PRINT followed by a print list, then type OF followed by the RSE:

```
PRINT print-list OF an-rse
```

- If you want to name items subordinate to a list (OCCURS) field in the record, you can do this in one of several ways:

```
FOR an-rse
FOR a-list-field
PRINT print-list
```

```
FOR an-rse
PRINT ALL print-list OF list-field
```

```
PRINT ALL ALL print-list OF list-field OF an-rse
```

There are times when you can get by with fewer than the indicated number of ALLs or put them in different positions in the print list. To be safe, put in an ALL for each OF clause in the PRINT statement. To make it simple, put the ALLs before the print list.

The most difficult thing to do in a PRINT statement is to specify a print list that includes only certain list items subordinate to an OCCURS clause. Here is an example that uses the view domain EMPLOYEE_HISTORY_2 (discussed in Chapter 15) to display only one field subordinate to each OCCURS clause. The PRINT statement indents lines to show the list items and their corresponding repeating field defined with an OCCURS clause:

```
DTR> READY EMPLOYEE_HISTORY_2
DTR> PRINT ALL LAST_NAME,
CON>     ALL JOB_START, !Subordinate to JOBS_HERE
CON>     ALL SALARY_AMOUNT - !Subordinate to
CON>                                     !SALARIES_FOR_JOBS
CON>     OF SALARIES_FOR_JOBS -
CON>     OF JOBS_HERE -
CON>     OF EMPLOYEE_HISTORY_2 WITH EMPLOYEE_ID = "00168"
```

LAST NAME	JOB START	SALARY AMOUNT
Nash	23-Feb-1979	\$27,126.00
		\$25,057.00
		\$23,919.00
		\$23,605.00
		\$21,520.00
30-Oct-1977	1-Jul-1975	\$20,883.00
		\$15,977.00
		\$15,851.00
		\$15,179.00

```
DTR>
```

Table 19-1 lists and describes the elements you can include in a print list.

Table 19-1: Print List Elements

Print List Element	Function and Results
field-name [modifier]	Specifies the field whose contents are to be formatted and printed. The optional modifier describes the column header for the field or the format of the output or both. (See Table 19-2.) If the field is a group field, DATATRIEVE displays all the elementary fields contained in that group field. If you omit this element, all the elementary fields in a record are displayed.
literal [modifier] *.prompt-name [modifier] **.prompt-name [modifier] arithmetic-exp [modifier] statistical-exp [modifier]	Specifies a value expression to be evaluated and printed. The optional modifier describes the column header for the value expression or the format of the display, or both. (See Table 19-2.) Chapter 18 discusses these value expressions.
SPACE [n]	Inserts n horizontal spaces before the next print list element. If you omit n, DATATRIEVE inserts one space before the next print list element.
TAB [n]	Inserts the space of n tab characters before the next print list element. If you omit n, DATATRIEVE inserts the space of one tab before the next print list element. DATATRIEVE assumes that tabs are set every eight spaces and inserts enough spaces (not actually tab characters) in the print line to start the next print list element in the appropriate column.
COL n	Specifies that the following print list element begins in column n of the display line. If n is less than the current column number, DATATRIEVE skips a line and begins the next print list element in column n. The first column in the line is column 1.
SKIP [n]	Begins the output of the next print list element at the beginning of the nth line from the current line. If n is greater than 1, the intervening lines are blank. If you omit n, DATATRIEVE moves the cursor to the beginning of the next line. If you omit this print list element, DATATRIEVE displays multilined output on consecutive lines.
NEW_PAGE	Moves the cursor to the top of a new print page. Column headers are suppressed, and output begins at column 1 unless another print list element changes the position of the cursor.

19.2.2 Using Print List Modifiers

Table 19-2 describes the options you use to control a header above each value you are printing and to specify an edit string to format that value. If you want to specify both a header and an edit string for a value, enter the header modifier first.

Table 19-2: Print Item Modifiers

Modifier	Effect
("column-header")	Specifies a column header for the preceding data item. You must enclose column header text in double quotation marks and the entire modifier in parentheses. To specify a multiline column header, separate one segment of the column header from another with a slash (/), as in ("WHOLE"/"NAME"). When you use this modifier in your PRINT statement, it overrides a header specified in a QUERY_HEADER clause when the field was defined.
(-)	Suppresses the printing of a column header for the preceding data item. You must enclose the hyphen in parentheses.
USING edit-string	Tells DATATRIEVE how to format the preceding data item. The specified edit string overrides any edit string specified in an EDIT_STRING clause when a field was defined. See the section on the EDIT_STRING clause in Chapter 9 for information on forming edit strings. If you follow an edit string with another print list element, include a space between the last character of the edit string and the comma that separates one print list element from another. Otherwise, DATATRIEVE considers the comma part of the edit string.

19.2.3 Sending Output to a File or Printer

There are two ways you can specify where you want PRINT statement output to go:

- You can precede the PRINT statement with one or more ON statements.
- You can append an ON clause to the end of the PRINT statement.

If you do not specify an ON statement or ON clause, DATATRIEVE sends the output of the PRINT statement to the device assigned to the logical name SYS\$OUTPUT. Usually, this is your terminal (TT:).

As the following examples illustrate, using the ON statement gives you the options you do not have when you use the ON clause. With the ON statement, you can send the output of more than one PRINT statement to the same device or file. You can also send the output of one or more PRINT statements to more than one device or file:

```
ON LP:
  BEGIN
    PRINT...
    PRINT...
  END

ON DBA2:[PERSONNEL.REPORTS]SALARIES.DAT
ON TT:
  PRINT...

PRINT... ON SALARIES.DAT
```

If, on your system, the term "LP:" is defined to mean something other than a line printer, the statement ON LP: might not work for you. In this case, you have to specify a particular device, such as LQP or LPA0:

You can specify a prompting value expression in place of a file specification in either the ON statement or the ON clause. You can therefore write procedures that let users specify where they want output printed or filed:

```
ON *."enter device or file name"
  BEGIN
    PRINT...
    PRINT...
  END
```

Writing Reports 20

This chapter tells you how to use the VAX DATATRIEVE Report Writer. For a more comprehensive discussion of the Report Writer, refer to the *VAX DATATRIEVE Guide to Writing Reports*.

The SUM statement and the PRINT statement (discussed in Chapters 18 and 19) give you control over the display of your data. The DATATRIEVE Report Writer can do more formatting for you than either the SUM statement or the PRINT statement alone can do. The Report Writer can:

- Center a report name at the top of the page
- Print the current date at the upper right
- Print page numbers at the upper right
- Set up column headings
- Print a detail line for each record
- Print summary lines for selected groups of records and for the entire report

You create a DATATRIEVE report with a series of Report Writer statements called a **report specification**. A report specification controls the format and determines the content of your report. Some Report Writer statements are required for a valid report specification and others are optional.

Example 20-1 shows a sample report specification and the report it creates. Subsequent sections in this chapter show you how to use Report Writer options to improve the report's format and to increase the information it can produce from the data on which it is based. Comment lines begin with an exclamation point (!) and point out features in the input that follows.

Example 20-1: Sample Report

```
DTR> ! Before you invoke the Report Writer, ready the domains or
DTR> ! views containing your data. If you prefer, you can create
DTR> ! a collection on which to base your report.
DTR> !
DTR> READY CURRENT_JOBS, CURRENT_SALARIES
DTR> !
DTR> ! The REPORT statement invokes the Report Writer and
DTR> ! specifies the data on which you want to base the report.
DTR> ! In this case, the data specification is an RSE that
DTR> ! joins records in two views. The RW> prompt indicates
DTR> ! that the Report Writer is waiting for your input.
DTR> !
DTR> REPORT CURRENT_JOBS CROSS CURRENT_SALARIES OVER
RW>   EMPLOYEE_ID WITH DEPARTMENT_CODE = "ELEL"
RW> !
RW> ! If you want your report to have a name, you must
RW> ! specify one. The SET REPORT_NAME statement lets you
RW> ! specify a name for your report that will be centered on
RW> ! the first line (in this case, on the first two lines).
RW> ! Note that the way you enter a report name follows the
RW> ! same conventions you use for entering column headers and
RW> ! text literals, covered in Chapter 19.
RW> !
RW> SET REPORT_NAME = "SALARIES IN"/"ELECTRICAL -
CON> ENGINEERING DEPARTMENT"
RW> !
RW> ! You specify one PRINT statement in a report. It lists
RW> ! the items you want to display in each detail line. It
RW> ! can also specify any edit strings and column headers you
RW> ! might want for items in the print list. The following
RW> ! PRINT statement specifies only three fields and lets
RW> ! DATATRIEVE use the default edit strings and column
RW> ! headers that apply to those fields.
RW> !
RW> PRINT JOB_CODE, EMPLOYEE_ID, SALARY_AMOUNT
RW> !
RW> ! You can use the AT BOTTOM statement when you want to
RW> ! summarize data for a group of detail lines. The
RW> ! following AT BOTTOM statement calculates and displays
RW> ! the total value for all the salaries in the report.
RW> !
RW> AT BOTTOM OF REPORT PRINT TOTAL SALARY_AMOUNT USING
RW>   $$$,$$$,$$$$.99
RW> !
RW> ! You must finish a report with the END_REPORT statement.
RW> ! Following END_REPORT, DATATRIEVE displays the report
RW> ! and returns you to the DTR> prompt.
RW> !
RW> END_REPORT
DTR>
```

SALARIES IN
ELECTRICAL ENGINEERING DEPARTMENT

10-Apr-1984
Page 1

JOB CODE	EMPLOYEE ID	SALARY AMOUNT
SANL	00172	\$55,413.00
SPGM	00206	\$29,692.00
SANL	00211	\$42,554.00
GFER	00222	\$14,122.00
GFER	00231	\$10,907.00
EENG	00238	\$24,330.00
GFER	00240	\$10,188.00
SANL	00273	\$44,264.00
MENG	00296	\$20,770.00
APGM	00377	\$15,646.00
MENG	00393	\$32,558.00
EENG	00428	\$27,363.00
JNTR	00443	\$10,232.00
MENG	00458	\$22,690.00
SPGM	00460	\$25,000.00
GFER	00461	\$10,364.00
PRGM	00480	\$26,032.00
MENG	00489	\$32,488.00
		\$454,613.00

DTR> ! If you plan to produce the report frequently, you can put
DTR> ! the commands and statements you need to produce it in a
DTR> ! procedure. You can then simply execute the procedure every
DTR> ! time you want the report. Refer to Chapter 17 for details
DTR> ! on creating procedures.

```
DTR>
DTR> SHOW SALARIES_REPORT
PROCEDURE SALARIES_REPORT
!
READY CURRENT_JOBS SHARED READ, CURRENT_SALARIES SHARED READ
!
REPORT CURRENT_JOBS CROSS CURRENT_SALARIES OVER
EMPLOYEE_ID WITH DEPARTMENT_CODE = "ELEL"
!
SET REPORT_NAME = "SALARIES IN"/"ELECTRICAL ENGINEERING DEPARTMENT"
!
PRINT JOB_CODE, EMPLOYEE_ID, SALARY_AMOUNT
!
AT BOTTOM OF REPORT PRINT TOTAL SALARY_AMOUNT USING
    $$$,$$$,$$$ .99
!
END_REPORT
!
FINISH CURRENT_JOBS, CURRENT_SALARIES
!
END_PROCEDURE

DTR>
```

The following sections more fully discuss each part of a report specification.

20.1 Entering the REPORT Statement

A report specification must begin with the keyword REPORT. Unless you are reporting on all records in the CURRENT collection in the order that they appear, you then enter an RSE to tell DATATRIEVE the location of the records containing the data you are using, which records you want, and how you want the records sorted. Refer to the sections in Chapters 14 and 15 that discuss and illustrate RSEs if you need more information about creating RSEs.

A REPORT statement can also specify an ON clause to direct report output to a specific device or file. If you want to direct report output to more than one device or file, you can put your entire report specification after more than one ON statement. If you do not specify either an ON clause or an ON statement, DATATRIEVE displays the report on your terminal.

Here are some sample REPORT statements:

```
DTR> REPORT
RW> !
RW> ! Uses all the records in the CURRENT collection in default
RW> ! sort order.
```

```
DTR> REPORT CURRENT SORTED BY DEPARTMENT_CODE, JOB_CODE ON
CON> LP:
RW> !
RW> ! Uses all the records in the CURRENT collection, but first
RW> ! sorts them according to DEPARTMENT_CODE and then,
RW> ! within each DEPARTMENT_CODE group, according to
RW> ! JOB_CODE. Sends the report to the system line printer.
```

```
DTR> REPORT CURRENT_JOBS WITH DEPARTMENT_CODE = "ADMN"
RW> !
RW> ! Uses only those records in CURRENT_JOBS that contain
RW> ! the specified DEPARTMENT_CODE value and uses them in
RW> ! default sort order.
```

```

DTR> REPORT CURRENT_JOBS CROSS CURRENT_SALARIES OVER
CON>   EMPLOYEE_ID WITH DEPARTMENT_CODE = *."department code"
RW>   !
RW>   ! Uses records that combine data from CURRENT_JOBS and
RW>   ! CURRENT_SALARIES, but only those containing the
RW>   ! DEPARTMENT_CODE value entered in response to the prompt.
RW>   ! Detail lines for the records will appear in default
RW>   ! sort order.

```

```

DTR> ON SALARIES.RPT
DTR> ON TT:
DTR> REPORT ...
RW>   !
RW>   ! Writes the report to the file SALARIES.RPT in your
RW>   ! current VMS directory and also displays the report
RW>   ! on your terminal.

```

After the REPORT statement, the only remaining requirement for your report specification is the END_REPORT statement. Usually, however, you want to enter one or more SET statements to control report headings, report width, and page size. (A PRINT statement, while not required in a report, is needed to specify detail lines.)

20.2 Controlling Headers and Other Report Settings

Use one or more Report Writer SET statements to specify:

- What you want the report header to look like, if you do not want the format DATATRIEVE uses by default
- The width of the report in columns per page, if you do not want the 80-column default
- The number of lines you want on each page of the report, if you do not want the 60-line default
- The maximum number of lines or pages you want the report to contain, if you want to set a limit to the size of your report

Table 20-1 lists the elements you can include in a SET statement and explains what each can do for you. The first column indicates the general objective for the setting.

The next three columns indicate the keywords that can follow SET, the function of the statement, and the Report Writer's default setting if the statement is not used.

The Prompt Option column indicates whether or not a prompting value expression may be included with a form of the SET statement. If you include a prompt, the Report Writer prompts you for a value when it processes the report specification.

The Maximum Value column indicates the largest value you can specify for a form of the SET statement.

Table 20-1: Report Writer SET Statement Options

Setting For	Keywords	Function	Default	Prompt Option	Maximum Value
Report Header	REPORT_NAME=	Specifies a name for the report and centers the name on the first line of each page	No report name	Yes; response to prompt must be enclosed in quotation marks	-
	DATE=	Specifies a date or string and prints it on the upper right line of each page	Current system date	No	-
	NO DATE	Suppresses the printing of a date	Current system date	No	-
	NUMBER=	Causes the Report Writer to print the specified page number below the date	Current page number	Yes	99,999
	NO NUMBER	Suppresses the printing of a page number	Current page number	No	-
	NO REPORT_HEADER	Suppresses the printing of report name, date, column headers, and page number on each page	Header printed on each page of the report	No	-

(continued on next page)

Table 20-1: Report Writer SET Statement Options (Cont.)

Setting For	Keywords	Function	Default	Prompt Option	Maximum Value
Column Headers	NO COLUMN_HEADER	Suppresses the printing of column headers	Headers printed on each page	No	-
Page Size	COLUMNS_PAGE=	Specifies the page width in columns	Current terminal width or 80 columns	Yes	255
	LINES_PAGE=	Specifies the page length in lines	60 lines	Yes	About 2 billion
Report Size	MAX_LINES=	Specifies the maximum lines for the report	No limit	Yes	About 2 billion
	MAX_PAGES=	Specifies the maximum pages for the report	No limit	Yes	About 2 billion

Example 20-2 illustrates how different combinations of SET statements in the report specification vary the format of the report produced in Example 20-1.

Example 20-2: Using SET Statements to Vary Report Format

```
DTR> SHOW SALARIES_REPORT
PROCEDURE SALARIES_REPORT
READY CURRENT_JOBS, CURRENT_SALARIES SHARED READ
!
REPORT CURRENT_JOBS CROSS CURRENT_SALARIES OVER
  EMPLOYEE_ID WITH DEPARTMENT_CODE = "ELEL"
!
SET REPORT_NAME = "SALARIES IN"/"ELECTRICAL ENGINEERING DEPARTMENT"
SET COLUMNS_PAGE = 50
SET DATE = "Wednesday, April 11"
SET NO NUMBER
!
PRINT JOB_CODE, EMPLOYEE_ID, SALARY_AMOUNT
!
AT BOTTOM OF REPORT PRINT TOTAL SALARY_AMOUNT USING
  $$$,$$$,$$$ .99
!
END_REPORT
```

(continued on next page)

!
FINISH CURRENT_JOBS, CURRENT_SALARIES
END_PROCEDURE
DTR) :SALARIES_REPORT

SALARIES IN
ELECTRICAL ENGINEERING DEPARTMENT
Wednesday, April 11

JOB CODE	EMPLOYEE ID	SALARY AMOUNT
SANL	00172	\$55,413.00
SPGM	00206	\$29,692.00
SANL	00211	\$42,554.00
GFER	00222	\$14,122.00
GFER	00231	\$10,907.00

. . .
. . .
. . .

DTR) EDIT SALARIES_REPORT

. . .
. . .
. . .

DTR) SHOW SALARIES_REPORT

. . .
. . .
. . .

SET REPORT_NAME = "SALARIES IN"/"ELECTRICAL ENGINEERING"/"DEPARTMENT"
SET COLUMNS_PAGE = 70
SET DATE = "RESTRICTED DISTRIB."

. . .
. . .
. . .

DTR) :SALARIES_REPORT

SALARIES IN
ELECTRICAL ENGINEERING
DEPARTMENT

RESTRICTED DISTRIB.
Page 1

JOB CODE	EMPLOYEE ID	SALARY AMOUNT
SANL	00172	\$55,413.00
SPGM	00206	\$29,692.00
SANL	00211	\$42,554.00
GFER	00222	\$14,122.00
GFER	00231	\$10,907.00
EENG	00238	\$24,330.00

DTR> EDIT SALARIES_REPORT

DTR> SHOW SALARIES_REPORT

SET COLUMNS_PAGE = 60
SET NO REPORT_HEADER

DTR> SALARIES_REPORT

JOB CODE	EMPLOYEE ID	SALARY AMOUNT
SANL	00172	\$55,413.00
SPGM	00206	\$29,692.00
SANL	00211	\$42,554.00
GFER	00222	\$14,122.00
GFER	00231	\$10,907.00
EENG	00238	\$24,330.00

DTR>

20.3 Specifying Detail Lines

Detail lines usually take up the most space in your report. A detail line represents the set of values contributed by an individual record. Use one `PRINT` statement in your report specification to tell `DATATRIEVE`:

- The fields from each record that you want to print in each detail line
- Other fields that you want to print in each detail line
- Value expressions that you want to print in each detail line
- How to space the items in a detail line across the report page

The following sections explain these options. Example 20-3 illustrates various formats and contents for one detail line.

20.3.1 Specifying and Formatting Values in a Detail Line

Most often you specify record field names to represent values in a detail line. Sometimes you may want to specify names of variables you have declared or other value expressions (for example, `SALARY_AMOUNT - DEDUCTIONS`).

When you want to display value expressions that are computed by one or more fields, you might want to include a column header and edit string to display the results in that column of data. You might also want to include column headers and edit strings for field values to override the `DATATRIEVE` defaults. If you need to review how to handle column headers and edit strings, refer to the section on the `PRINT` statement in Chapter 19.

20.3.2 Spacing Values in a Detail Line Across the Page

By default, `DATATRIEVE` spaces the items in your detail lines evenly across the number of columns set for the report page. You might want to vary this spacing by including the `PRINT` list elements `COL n` and `SPACE n` in your detail lines. The table in Chapter 19 on print list elements explains these options.

Example 20-3: Varying the Format of Detail Lines

```

DTR> SHOW SALARIES_REPORT
PROCEDURE SALARIES_REPORT
READY CURRENT_JOBS, CURRENT_SALARIES SHARED READ
!
REPORT CURRENT_JOBS CROSS CURRENT_SALARIES OVER
  EMPLOYEE_ID WITH DEPARTMENT_CODE = "ELEL"
!
SET COLUMNS_PAGE = 70
SET REPORT_NAME = "SALARIES FOR"/"ELECTRICAL ENGINEERING"
!
PRINT JOB_CODE, EMPLOYEE_ID ("ID"), SPACE 1, EMPLOYEE_ID VIA
WHO_IS_IT ("NAME"), SALARY_AMOUNT
!
AT BOTTOM OF REPORT PRINT TOTAL SALARY_AMOUNT USING
  $$$, $$$, $$$ .99
!
END_REPORT
!
FINISH CURRENT_JOBS, CURRENT_SALARIES
END_PROCEDURE

```

DTR> :SALARIES_REPORT

SALARIES FOR
ELECTRICAL ENGINEERING

11-Apr-1984
Page 1

JOB CODE	ID	EMPLOYEE NAME	SALARY AMOUNT
SANL	00172 Peters	Janis K	\$55,413.00
SPGM	00206 Stornelli	Marty J	\$29,692.00
SANL	00211 Gutierrez	Ernest F	\$42,554.00
GFER	00222 Lasch	Norman	\$14,122.00
GFER	00231 Clairmont	Rick	\$10,907.00
.	.	.	.
.	.	.	.
.	.	.	.

DTR> EDIT SALARIES_REPORT

DTR> SHOW SALARIES_REPORT
PROCEDURE SALARIES_REPORT

(continued on next page)


```

!
DECLARE NAME COMPUTED BY EMPLOYEE_ID VIA WHO_IS_IT
  QUERY_HEADER IS "NAME OF EMPLOYEE"
  EDIT_STRING IS X(25).
!
REPORT CURRENT_JOBS CROSS CURRENT_SALARIES OVER
  EMPLOYEE_ID WITH DEPARTMENT_CODE = "ELEL"
!
SET COLUMNS_PAGE = 65
SET REPORT_NAME = "SALARIES FOR"/"ELECTRICAL ENGINEERING"
!
PRINT COL 8, JOB_CODE, SPACE 2, EMPLOYEE_ID ("ID"), SPACE 2, NAME,
  SPACE 3, SALARY_AMOUNT USING $$$$,$$$$.99

```

DTR) :SALARIES_REPORT

SALARIES FOR
ELECTRICAL ENGINEERING

11-Apr-1984
Page 1

JOB CODE	ID	NAME OF EMPLOYEE			SALARY AMOUNT
SANL	00172	Peters	Janis	K	\$55,413.00
SPGM	00206	Stornelli	Marty	J	\$29,692.00
SANL	00211	Gutierrez	Ernest	F	\$42,554.00
GFER	00222	Lasch	Norman		\$14,122.00
GFER	00231	Clairmont	Rick		\$10,907.00

DTR)

20.4 Handling Control Groups

Often you need to report not only on a whole body of data but also on groups of records within it. A series of sorted records that have the same value in one or more fields or the same computed value is called a **control group**. If the SALARIES_REPORT procedure used in this chapter generated a report that summarized salary data for each DEPARTMENT_CODE, rather than only one of them, then all the records having the same value in the DEPARTMENT_CODE field would form a control group.

Reports can contain more than one control group. SALARIES_REPORT could also summarize salary data for each JOB_CODE value within each group having the same DEPARTMENT_CODE value. In this case, SALARIES_REPORT would have two control groups.

20.4.1 Sorting Records According to Control Group Key Values

The most important thing to remember when including control groups in a report is that the records must be sorted according to the control group values you want to use. Sometimes records from a domain, view, or collection are already sorted the way you want them to appear in the report. When they are not, you must include a SORTED BY clause in the REPORT statement RSE to get the records in the proper order.

To revise the SALARIES_REPORT procedure so that it generates a report to summarize salary data for each department and for each type of job within each department, the REPORT statement would need to look like this:

```
REPORT CURRENT_JOBS CROSS CURRENT_SALARIES OVER  
  EMPLOYEE_ID SORTED BY DEPARTMENT_CODE, JOB_CODE
```

20.4.2 Printing Control Group Headers

To print a header for a series of records within a control group, use the AT TOP OF statement in your report specification:

```
AT TOP OF DEPARTMENT_CODE PRINT DEPARTMENT_CODE  
AT TOP OF JOB_CODE PRINT JOB_CODE
```

20.4.3 Printing Control Group and Report Summaries

To print summary information for a series of records within a control group, use the AT BOTTOM OF statement:

```
AT BOTTOM OF DEPARTMENT_CODE PRINT TOTAL SALARY_AMOUNT  
AT BOTTOM OF JOB_CODE PRINT AVERAGE SALARY_AMOUNT
```

You might want to print some sort of explanation for the summary values you are computing. In this case, declare a COMPUTED BY variable to store the summary value you want to display. Displaying the variable value, rather than the expression directly, ensures that the explanation for the value and the value itself are printed on the same line of the report:

```
DECLARE TOTAL_SALARY COMPUTED BY TOTAL SALARY_AMOUNT
          EDIT_STRING $$$,$$$,$$$ .99.
DECLARE AVERAGE_SALARY COMPUTED BY AVERAGE SALARY_AMOUNT
          EDIT_STRING $$$,$$$ .99.
```

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

```
AT BOTTOM OF DEPARTMENT_CODE
  PRINT "TOTAL SALARIES FOR DEPT: ", TOTAL_SALARY (-)
AT BOTTOM OF JOB_CODE
  PRINT "AVERAGE SALARY FOR JOB: ", AVERAGE_SALARY (-)
```

Example 20-4 shows the SALARIES_REPORT procedure modified to include control groups. The example uses the keywords COL, SKIP, and NEW_PAGE to control the position of the values immediately following them in the print list. The example also includes field names in concatenation value expressions to make the control group summaries more descriptive. Refer to Chapter 19 if you need to review how to use concatenation characters and print list keywords.

Example 20-4: Including Control Groups in a Report

```
DTR> SHOW SALARIES_REPORT
PROCEDURE SALARIES_REPORT
READY CURRENT_JOBS, CURRENT_SALARIES SHARED READ
!
DECLARE NAME COMPUTED BY EMPLOYEE_ID VIA WHO_IS_IT
  QUERY_HEADER IS "NAME OF EMPLOYEE"
  EDIT_STRING IS X(25).
DECLARE TOTAL_SALARY COMPUTED BY TOTAL SALARY_AMOUNT
  EDIT_STRING IS $$$,$$$,$$$ .99.
DECLARE AVERAGE_SALARY COMPUTED BY AVERAGE SALARY_AMOUNT
  EDIT_STRING IS $$$,$$$ .99.
!
REPORT CURRENT_JOBS CROSS CURRENT_SALARIES OVER
  EMPLOYEE_ID SORTED BY DEPARTMENT_CODE, JOB_CODE
!
SET COLUMNS_PAGE = 65
SET REPORT_NAME = "SALARY DATA"/"BY DEPARTMENTS"
!
PRINT EMPLOYEE_ID ("ID"), NAME,
  SALARY_AMOUNT USING $$$,$$$,$$$ .99
```

```

!
AT TOP OF DEPARTMENT_CODE PRINT DEPARTMENT_CODE ("DEPT")
AT TOP OF JOB_CODE PRINT JOB_CODE
AT BOTTOM OF JOB_CODE
  PRINT COL 54, "-----", SKIP, COL 27,
  "AVERAGE SALARY FOR"!!!!JOB_CODE!":", AVERAGE_SALARY (-), SKIP
AT BOTTOM OF DEPARTMENT_CODE
  PRINT COL 51, "-----", SKIP, COL 24,
  "TOTAL SALARIES FOR"!!!!DEPARTMENT_CODE!":",
  TOTAL_SALARY (-), NEW_PAGE
!
AT BOTTOM OF REPORT
  PRINT COL 51, "-----", SKIP, COL 26,
  "GRAND TOTAL SALARIES: ", TOTAL_SALARY (-)
!
END_REPORT
!
FINISH CURRENT_JOBS, CURRENT_SALARIES
END_PROCEDURE
DTR> :SALARIES_REPORT

```

SALARY DATA
BY DEPARTMENTS

12-Apr-1984
Page 1

DEPT	JOB CODE	ID	NAME OF EMPLOYEE			SALARY AMOUNT
ADMN	DSUP					
		00472	Delano	Al	F	\$39,531.00
			AVERAGE SALARY FOR DSUP:			\$39,531.00
	EENG					
		00188	Clarke	Karen	G	\$21,093.00
		00300	Gramby	Marjorie		\$23,856.00
			AVERAGE SALARY FOR EENG:			\$22,474.50
	JNTR					
		00330	Williams	Christine B		\$15,694.00
			AVERAGE SALARY FOR JNTR:			\$15,694.00
	MENG					
		00438	Wilkins	Mark		\$20,589.00
		00190	O'Sullivan	Rick	G	\$34,976.00
			AVERAGE SALARY FOR MENG:			\$27,782.50
	PRSD					
		00225	Jackson	Mary Lou		\$8,687.00
			AVERAGE SALARY FOR PRSD:			\$8,687.00

(continued on next page)

```

SPGM
00488  McGrath      Tom           $26,334.00
-----
                AVERAGE SALARY FOR SPGM:  $26,334.00

VPSD
00204  Myotte      Charles K    $87,143.00
00228  Harrison   Lisa        $85,150.00
00439  Smoot      Mary Lou    $83,199.00
00471  Herbener   James Q     $83,905.00
00435  MacDonald  Johanna P   $84,147.00
00267  Saninocencio Roger H     $80,812.00
00271  Gramby     Karen Z     $75,113.00
00359  Crain      Jesse      $93,340.00
00494  Raiola-Paul Barbara     $78,660.00
00415  Mistretta  Kathleen G  $86,124.00
-----
                AVERAGE SALARY FOR VPSD:  $83,759.30

                -----
TOTAL SALARIES FOR ADMN:  $1,028,353.00
.
.
.
.
.
.
                -----
GRAND TOTAL SALARIES:  $8,967,907.00

```

DTR>

20.5 Including a Title Page for the Report

If you want to include a title page for your report, use the `AT TOP OF REPORT` statement. Example 20-5 shows `SALARIES_REPORT` modified to include a title page. Note that `NEW_PAGE` is the last item in the `AT TOP OF REPORT` print list. This ensures that the main body of the report does not start to print on the title page. The report specification sets the first page number to 2. (Page numbering is suppressed for the title page itself.)

Example 20-5: Creating a Title Page

```
DTR> SHOW SALARIES_REPORT
PROCEDURE SALARIES_REPORT
READY CURRENT_JOBS, CURRENT_SALARIES SHARED READ
!
DECLARE NAME COMPUTED BY EMPLOYEE_ID VIA WHO_IS_IT
  QUERY_HEADER IS "NAME OF EMPLOYEE"
  EDIT_STRING IS X(25).
DECLARE TOTAL_SALARY COMPUTED BY TOTAL SALARY_AMOUNT
  EDIT_STRING IS $$, $$$, $$$ .99.
DECLARE AVERAGE_SALARY COMPUTED BY AVERAGE SALARY_AMOUNT
  EDIT_STRING IS $$$, $$$ .99.
DECLARE DATE_TODAY USAGE DATE
  DEFAULT VALUE IS "TODAY".
!
REPORT CURRENT_JOBS CROSS CURRENT_SALARIES OVER
  EMPLOYEE_ID SORTED BY DEPARTMENT_CODE, JOB_CODE
!
AT TOP OF REPORT PRINT SKIP 15, COL 27, "SALARY DATA",
  SKIP, COL 25, "BY DEPARTMENTS", SKIP 5, COL 27,
  DATE_TODAY (-), SKIP 15, COL 21,
  "RESTRICTED DISTRIBUTION", NEW_PAGE
!
SET COLUMNS_PAGE = 65
SET NO DATE
SET NUMBER = 2
!
PRINT EMPLOYEE_ID ("ID"), NAME,
  SALARY_AMOUNT USING $$$$, $$$ .99
!
AT TOP OF DEPARTMENT_CODE PRINT DEPARTMENT_CODE ("DEPT")
AT TOP OF JOB_CODE PRINT JOB_CODE
AT BOTTOM OF JOB_CODE
  PRINT COL 54, "-----", SKIP, COL 27,
    "AVERAGE SALARY FOR"!!!JOB_CODE!":", AVERAGE_SALARY (-),
    SKIP
AT BOTTOM OF DEPARTMENT_CODE
  PRINT COL 51, "-----", SKIP, COL 24,
    "TOTAL SALARIES FOR"!!!DEPARTMENT_CODE!":",
    TOTAL_SALARY (-), NEW_PAGE
!
AT BOTTOM OF REPORT
  PRINT COL 51, "-----", SKIP, COL 26,
    "GRAND TOTAL SALARIES: ", TOTAL_SALARY (-)
!
END_REPORT
!
FINISH CURRENT_JOBS, CURRENT_SALARIES
END_PROCEDURE

DTR> :SALARIES_REPORT
```

(continued on next page)

SALARY DATA
BY DEPARTMENTS

12-Apr-1984

RESTRICTED DISTRIBUTION

DEPT	JOB CODE	ID	NAME OF EMPLOYEE			SALARY AMOUNT
ADMN	DSUP	00472	Delano	A1	F	\$39,531.00
			AVERAGE SALARY FOR DSUP:			\$39,531.00
	EENG	00188	Clarke	Karen	G	\$21,093.00
	
	
	

DTR>

20.6 Exiting the Report Writer and Correcting Mistakes

You normally exit the Report Writer by entering the `END_REPORT` statement. `DATATRIEVE` processes the report specification and produces the report if there are no errors.

If `DATATRIEVE` detects an error in your Report Writer statements, it displays an error message and returns you to `DATATRIEVE` command level. You can immediately type `EDIT` to correct your mistakes. Your editing buffer will contain your entire report specification (or your entire procedure if you are creating the report specification in a procedure).

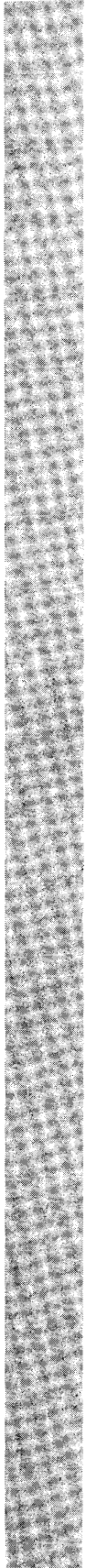
To force an exit from the Report Writer and return to `DATATRIEVE` command level, you can enter `CTRL/C` as a response to an `RW>` prompt or in the middle of an input line:

```
DTR> READY CURRENT_SALARIES
DTR> REPORT CURRENT_SALARIES SORTED BY JOB_CODE
RW> CTRL/C
^C
Execution terminated by operator
DTR>
```




Part VII
Appendixes and Index





DATATRIEVE Keywords **A**

This appendix contains the keywords and function names that you should not use when you are naming items.

The following table lists all VAX DATATRIEVE keywords.

Table A-1: DATATRIEVE Keywords

* (asterisk)) (right parenthesis)	APPLICATION_KEYPAD
@ (at sign)	; (semicolon)	ARGUMENTS
: (colon)	/ (slash)	AS
, (comma)	_ (underscore)	ASC
** (double asterisk)	(vertical bar)	ASCENDING
“ (double quotation mark)	ABORT	AT
= (equal sign)	ADT	AVERAGE
! (exclamation point)	ADVANCED	BANISH
> (greater than sign)	AFTER	BATCH
- (hyphen or minus sign)	ALIGNED_MAJOR_MINOR	BEFORE
((left parenthesis)	ALIN_MAJ_MIN	BEGIN
< (less than sign)	ALL	BETWEEN
. (period)	ALLOCATION	BLANK
+ (plus sign)	AND	BOOLEAN
? (question mark)	ANY	BOTTOM

(continued on next page)

Table A-1: DATATRIEVE Keywords (Cont.)

BT	CURRENCY	DOMAINS
BUT	CURRENT	DOUBLE
BY	DATABASE	DROP
BYTE	DATABASES	DUP
CHANGE	DATATYPE	EDIT
CHARACTER	DATE	EDIT_BACKUP
CHARACTERS	DEBUG	EDIT_STRING
CHOICE	DECIMAL	ELSE
CLOSE	DECLARE	END
COL	DECREASING	END_CHOICE
COLLECTIONS	DEFAULT	END_PLOT
COLUMN	DEFINE	END_PROCEDURE
COLUMN_HEADER	DEFINEP	END_REPORT
COLUMNS_PAGE	DELETE	END_TABLE
COMMIT	DELETEP	ENDING
COMP	DENY	ENTRY
COMP_1	DEPENDING	EQ
COMP_2	DESC	EQUAL
COMP_3	DESCENDING	ERASE
COMP_5	D_FLOATING	EXCLUSIVE
COMP_6	DICTIONARY	EXECUTE
COMPUTED	DICTIONARIES	EXIT
CONCURRENCY	DIGIT	EXTEND
CONNECT	DIGITS	EXTRACT
CONSISTENCY	DISCONNECT	F_FLOATING
CONT	DISPLAY	FIELDS
CONTAINING	DISPLAY_FORM	FILE
COUNT	DO	FILL
CROSS	DOMAIN	FILLER

(continued on next page)

Table A-1: DATATRIEVE Keywords (Cont.)

FIND	IS	MIN
FINISH	JUST	MISSING
FIRST	JUSTIFIED	MODIFY
FOR	JUSTIFY	NE
FORM	KEEP	NETWORK
FORMAT	KEY	NEW_PAGE
FORMS	KEYDEFS	NEW_SECTION
FROM	KEYWORD	NEXT
GE	LAST	NO
GET_FORM	LE	NONE
G_FLOATING	LEADING	NONLOCAL
GRANT	LEAVE	NOT
GREATER_EQUAL	LEFT	NOT_EQUAL
GREATER_THAN	LEFT_RIGHT	NOVERIFY
GROUP	LESS_EQUAL	NUMBER
GT	LESS_THAN	NUMERIC
GUIDE	LINES_PAGE	OCCURS
HELP	LIST	OCTA
HELP_LINES	LOCAL	OCTAWORD
HELP_PROMPT	LOCK_WAIT	OF
HELP_WINDOW	LONG	ON
H_FLOATING	LONGWORD	OPEN
IF	LT	OPTIMIZE
IN	MAJOR_MINOR	OR
INCR	MATCH	OVER
INCREASING	MAX	OVERPUNCHED
INIT_VECTOR	MAX_LINES	OWNER
INSERT	MAX_PAGES	PACKED
INTEGER	MEMBER	PAGE

(continued on next page)

Table A-1: DATATRIEVE Keywords (Cont.)

PATH	REDUCED	SORTED
PIC	RELEASE	SOURCE
PICTURE	REPEAT	SPACE
PLOT	REPORT	STARTING
PLOTS	REPORT_HEADER	STD_DEV
PORT	REPORT_NAME	STORE
PRINT	RETRIEVE	STRING
PRIOR	RIGHT	STRUCTURE
PRIVILEGES	ROLLBACK	SUBSCHEMA
PROCEDURE	RSE	SUM
PROCEDURES	RUNNING	SUPERCEDE
PROMPT	SCALE	SUPERSEDE
PROTECTED	SCHEMA	SYNC
PURGE	SCHEMAS	SYNCHRONIZED
PUT_FORM	SEARCH	SYNONYM
PW	SELECT	SYNONYMS
QUAD	SEMICOLON	TAB
QUADWORD	SEPARATE	TABLE
QUERY_HEADER	SET	TABLES
QUERY_NAME	SETS	TERMINAL
READ	SET_UP	TEXT
READY	SHARED	THE
REAL	SHOW	THEN
RECONNECT	SHOWP	TIMES
RECORD	SIGN	TO
RECORDS	SIGNED	TOP
RECOVER	SIZE	TOTAL
REDEFINE	SKIP	TRAILING
REDEFINES	SNAPSHOT	UIC
REDUCE	SORT	UNSIGNED

(continued on next page)

Table A-1: DATATRIEVE Keywords (Cont.)

USAGE	VARYING	WITH
USER	VECTOR	WITHIN
USING	VERIFY	WORD
VALID	VIA	WRITE
VALUE	WHEN	ZERO
VARIABLES	WHILE	ZONED

The following table lists all the DATATRIEVE functions. As with the keywords in the preceding table, you should not use function names when you are naming items.

Table A-2: DATATRIEVE Function Names

FN\$ABS	FN\$INIT_TIMER	FN\$SHOW_TIMER
FN\$ATAN	FN\$JULIAN	FN\$SIGN
FN\$COMMAND_KEYBOARD	FN\$KEYPAD_MODE	FN\$SIN
FN\$COS	FN\$KEYTABLE_ID	FN\$SPAWN
FN\$CREATE_LOG	FN\$LN	FN\$SQRT
FN\$DATE	FN\$LOAD_KEYDEFS	FN\$STR_EXTRACT
FN\$DAY	FN\$LOG10	FN\$STR_LOC
FN\$DCL	FN\$MINUTE	FN\$TAN
FN\$DEFINE_KEY	FN\$MOD	FN\$TIME
FN\$DELETE_KEY	FN\$MONTH	FN\$TODAY
FN\$DELETE_LOG	FN\$NINT	FN\$TRANS_LOG
FN\$EXP	FN\$OPENS_LEFT	FN\$UPCASE
FN\$FLOOR	FN\$PROMPT_KEYBOARD	FN\$VERSION
FN\$HEX	FN\$SECOND	FN\$WEEK
FN\$HOUR	FN\$SHOW_KEY	FN\$WIDTH
FN\$HUNDREDTH	FN\$SHOW_KEYDEFS	FN\$YEAR



Sample Record, Table, and View Definitions **B**

This appendix contains the record definitions for the domains in the sample personnel database used in this book. It also contains the table and view definitions that supplement or use the domains in that database.

Record definition for EMPLOYEES domain:

```
RECORD EMPLOYEES_REC USING
01 EMPLOYEES_REC.
    05 EMPLOYEE_ID          PIC X(5)
                           QUERY_NAME IS ID
                           QUERY_HEADER IS "ID".
    05 EMPLOYEE_NAME       PIC X(14)
                           QUERY_NAME IS NAME.
    10 LAST_NAME           PIC X(14)
                           QUERY_NAME IS L_NAME
                           QUERY_HEADER IS "LAST NAME".
    10 FIRST_NAME          PIC X(10)
                           QUERY_NAME IS F_NAME
                           QUERY_HEADER IS "FIRST NAME".
    10 MIDDLE_INITIAL      PIC X
                           QUERY_NAME IS INIT
                           QUERY_HEADER IS "INIT".
    05 EMPLOYEE_ADDRESS    PIC X(20)
                           QUERY_NAME IS ADDRESS.
    10 ADDRESS_DATA        PIC X(25).
    10 STREET              PIC X(25).
    10 TOWN                PIC X(20).
    10 STATE               PIC X(2).
    10 ZIP                 PIC X(5).
    05 SEX                 PIC X
                           VALID IF SEX = "M" OR SEX = "F".
    05 SOCIAL_SECURITY     PIC X(9)
                           EDIT_STRING IS XXXBXXBXXXX.
    05 BIRTHDAY            USAGE DATE
                           EDIT_STRING IS NN/DD/YY.
```

Record definition for JOB__HISTORY domain:

```
RECORD JOB_HISTORY_REC USING
01 JOB_HISTORY_REC.
   05 EMPLOYEE_ID          PIC X(5)
                           QUERY_NAME IS ID
                           VALID IF ID IN WHO_IS_IT.
   05 JOB_CODE             PIC X(4)
                           QUERY_NAME IS JOB
                           VALID IF JOB IN JOBS_TABLE.
   05 JOB_START            USAGE DATE.
   05 JOB_END              USAGE DATE
                           MISSING VALUE IS
                           "17-NOV-1858 00:00:00.00".
   05 DEPARTMENT_CODE     PIC X(4)
                           QUERY_NAME IS DEPT
                           VALID IF DEPT IN DEPARTMENTS_TABLE.
   05 SUPERVISOR_ID       PIC X(5)
                           QUERY_NAME IS SUPER
                           MISSING VALUE IS ""
                           VALID IF SUPER IN WHO_IS_IT OR
                           SUPER = "".
```

Record definition for SALARY__HISTORY domain:

```
RECORD SALARY_HISTORY_REC USING
01 SALARY_HISTORY_REC.
   05 EMPLOYEE_ID          PIC X(5)
                           QUERY_NAME IS ID
                           VALID IF ID IN WHO_IS_IT.
   05 SALARY_AMOUNT        USAGE LONG SCALE IS -2
                           QUERY_NAME IS SALARY
                           EDIT_STRING $$$,$$.99.
   05 SALARY_START         USAGE DATE.
   05 SALARY_END           USAGE DATE.
                           MISSING VALUE IS
                           "17-NOV-1858 00:00:00.00".
   05 REVIEW_CODE         PIC X.
```

Record definition for JOBS domain:

```
RECORD JOBS_REC USING
01 JOB.
   05 JOB_CODE             PIC X(4).
   05 MINIMUM_SALARY       USAGE LONG SCALE IS -2
                           EDIT_STRING $$$,$$.99.
   05 MAXIMUM_SALARY       USAGE LONG SCALE IS -2
                           EDIT_STRING $$$,$$.99.
   05 WAGE_CLASS           PIC X.
```

Record definition for DEGREES domain:

```
RECORD DEGREES_REC USING
01 DEGREES_REC.
   05 EMPLOYEE_ID          PIC X(5).
   05 COLLEGE_CODE         PIC X(4).
   05 DEGREE                PIC X(10).
   05 DEGREE_FIELD         PIC X(15).
   05 DATE_GIVEN           USAGE DATE.
;
```

Record definition for COLLEGES domain:

```
RECORD COLLEGES_REC
01 COLLEGE.
   05 COLLEGE_CODE         PIC X(4).
   05 COLLEGE_NAME         PIC X(25).
   05 CONTACT_NAME        PIC X(25).
   05 ADDRESS.
      10 ADDRESS_DATA     PIC X(20).
      10 STREET            PIC X(25).
      10 TOWN              PIC X(20).
      10 STATE             PIC XX.
      10 ZIP               PIC X(5).
;
```

Dictionary table DEPARTMENTS__TABLE:

```
TABLE DEPARTMENTS_TABLE
QUERY_HEADER IS "DEPARTMENT"/"NAME"
EDIT_STRING IS X(20)
"ADMN" : "Administration"
"ENG"  : "Engineering"
"MKTG" : "Marketing"
"MNFG" : "Manufacturing"
"PERS" : "Personnel"
"SALE" : "General Sales"
.      :
.      :
.      :
ELSE "Invalid Dept."
END_TABLE
```

Dictionary table JOBS__TABLE:

```
TABLE JOBS_TABLE
EDIT_STRING X(20)
QUERY_HEADER "JOB"/"TITLE"
"APGM" : "Associate Programmer",
"CLRK" : "Clerk",
"DGFR" : "Deputy Gopher",
"DMGR" : "Department Manager",
"DSUP" : "Dept. Supervisor",
"EEENG" : "Electrical Engineer",
"GFER" : "Gopher",
"JNTR" : "Janitor",
"MEENG" : "Mechanical Engineer"
"PRGM" : "Programmer"
"PRSD" : "Company President"
"SANL" : "Systems Analyst"
"SCTR" : "Secretary",
"SPGM" : "Systems Programmer",
"UPSD" : "Vice President"
ELSE "Not in JOBS_TABLE"
END_TABLE
```

Domain table WHO__IS__IT:

```
TABLE WHO_IS_IT FROM EMPLOYEES
QUERY_HEADER IS "          EMPLOYEE NAME          "
EDIT_STRING IS X(36)
USING EMPLOYEE_ID : EMPLOYEE_NAME
ELSE "ID not in EMPLOYEES."
END_TABLE
```

View ADDRESS__LIST:

```
DOMAIN ADDRESS_LIST OF EMPLOYEES, JOB_HISTORY USING
01 CURRENT_FOLKS OCCURS FOR EMPLOYEES CROSS
  JOB_HISTORY OVER EMPLOYEE_ID WITH JOB_END MISSING.
  03 LAST_NAME          FROM EMPLOYEES.
  03 FIRST_NAME        FROM EMPLOYEES.
  03 MIDDLE_INITIAL    FROM EMPLOYEES.
  03 ADDRESS_DATA      FROM EMPLOYEES.
  03 STREET            FROM EMPLOYEES.
  03 TOWN              FROM EMPLOYEES.
  03 STATE             FROM EMPLOYEES.
  03 ZIP              FROM EMPLOYEES.
;
```

View CURRENT__JOBS:

```
DOMAIN CURRENT_JOBS OF JOB_HISTORY
01 CURRENT_JOB OCCURS FOR JOB_HISTORY WITH
    JOB_END MISSING SORTED BY DEPARTMENT_CODE, JOB_CODE.
    03 DEPARTMENT_CODE      FROM JOB_HISTORY.
    03 JOB_CODE             FROM JOB_HISTORY.
    03 EMPLOYEE_ID         FROM JOB_HISTORY.
    03 JOB_START           FROM JOB_HISTORY.
    03 SUPERVISOR_ID       FROM JOB_HISTORY.
```

View CURRENT__SALARIES:

```
DOMAIN CURRENT_SALARIES OF SALARY_HISTORY
01 CURRENT_SALARY OCCURS FOR SALARY_HISTORY WITH
    SALARY_END MISSING.
    03 EMPLOYEE_ID         FROM SALARY_HISTORY.
    03 REVIEW_CODE        FROM SALARY_HISTORY.
    03 SALARY_AMOUNT      FROM SALARY_HISTORY.
```

View EMPLOYEE__HISTORY__1:

```
DOMAIN EMPLOYEE_HISTORY_1 OF EMPLOYEES, SALARY_HISTORY,
JOB_HISTORY USING
01 ONE_OF_US OCCURS FOR EMPLOYEES CROSS JOB_HISTORY OVER
    EMPLOYEE_ID WITH JOB_END MISSING.
    03 EMPLOYEE_ID      FROM EMPLOYEES.
    03 LAST_NAME        FROM EMPLOYEES.
    03 JOBS_HERE OCCURS FOR JOB_HISTORY WITH EMPLOYEE_ID =
        EMPLOYEES_REC.EMPLOYEE_ID SORTED BY
        DECREASING JOB_START.
        05 JOB_CODE      FROM JOB_HISTORY.
        05 DEPARTMENT_CODE FROM JOB_HISTORY.
        05 JOB_START     FROM JOB_HISTORY.
    03 SALARIES OCCURS FOR SALARY_HISTORY WITH
        (EMPLOYEE_ID = EMPLOYEES_REC.EMPLOYEE_ID) SORTED BY
        DECREASING SALARY_START.
        05 SALARY_START  FROM SALARY_HISTORY.
        05 SALARY_AMOUNT FROM SALARY_HISTORY.
```

View EMPLOYEE__HISTORY__2:

```
DOMAIN EMPLOYEE_HISTORY_2 OF EMPLOYEES, SALARY_HISTORY,  
JOB_HISTORY USING  
01 ONE_OF_US OCCURS FOR EMPLOYEES CROSS JOB_HISTORY OVER  
EMPLOYEE_ID WITH JOB_END MISSING.  
03 EMPLOYEE_ID FROM EMPLOYEES.  
03 LAST_NAME FROM EMPLOYEES.  
03 JOBS_HERE OCCURS FOR JOB_HISTORY WITH  
(EMPLOYEE_ID = EMPLOYEES_REC.EMPLOYEE_ID) AND  
(JOB_END NOT MISSING) SORTED BY DECREASING JOB_START.  
05 JOB_CODE FROM JOB_HISTORY.  
05 DEPARTMENT_CODE FROM JOB_HISTORY.  
05 JOB_START FROM JOB_HISTORY.  
05 SALARIES_FOR_JOBS OCCURS FOR SALARY_HISTORY WITH  
(EMPLOYEE_ID = JOB_HISTORY_REC.EMPLOYEE_ID) AND  
(SALARY_START BT JOB_START AND JOB_END) SORTED BY  
DECREASING SALARY_START.  
07 SALARY_START FROM SALARY_HISTORY.  
07 SALARY_AMOUNT FROM SALARY_HISTORY.  
;
```

View COLLEGE__LOCATIONS:

```
DOMAIN COLLEGE_LOCATIONS OF COLLEGES USING  
01 COLLEGE OCCURS FOR COLLEGES.  
03 COLLEGE_NAME FROM COLLEGES.  
03 TOWN FROM COLLEGES.  
03 STATE FROM COLLEGES.  
03 ZIP FROM COLLEGES.  
;
```

DATATRIEVE Sort Order **C**

The order and value associated with text characters is determined by the ASCII collating sequence. The following table lists characters in ascending order of ASCII values. Note that lowercase letters have higher values than uppercase letters.

Table C-1: DATATRIEVE Sort Order

Decimal Value	Character	Description
32		Space
33	!	Exclamation point
34	”	Double quotation mark
35	#	Number sign
36	\$	Dollar sign
37	%	Percent sign
38	&	Ampersand
39	'	Apostrophe, single quotation mark
40	(Left parenthesis
41)	Right parenthesis
42	*	Asterisk
43	+	Plus sign

(continued on next page)

Table C-1: DATATRIEVE Sort Order (Cont.)

Decimal Value	Character	Description
44	,	Comma
45	-	Minus sign, hyphen
46	.	Period, decimal point
47	/	Slash
48	0	Zero
49	1	One
50	2	Two
51	3	Three
52	4	Four
53	5	Five
54	6	Six
55	7	Seven
56	8	Eight
57	9	Nine
58	:	Colon
59	;	Semicolon
60	<	Less-than sign, left angle bracket
61	=	Equal sign
62	>	Greater-than sign, right angle bracket
63	?	Question mark
64	@	Commercial at sign
65	A	Uppercase A
66	B	Uppercase B
67	C	Uppercase C
68	D	Uppercase D
69	E	Uppercase E
70	F	Uppercase F

(continued on next page)

Table C-1: DATATRIEVE Sort Order (Cont.)

Decimal Value	Character	Description
71	G	Uppercase G
72	H	Uppercase H
73	I	Uppercase I
74	J	Uppercase J
75	K	Uppercase K
76	L	Uppercase L
77	M	Uppercase M
78	N	Uppercase N
79	O	Uppercase O
80	P	Uppercase P
81	Q	Uppercase Q
82	R	Uppercase R
83	S	Uppercase S
84	T	Uppercase T
85	U	Uppercase U
86	V	Uppercase V
87	W	Uppercase W
88	X	Uppercase X
89	Y	Uppercase Y
90	Z	Uppercase Z
91	[Left bracket
92	\	Backslash
93]	Right bracket
94	^	Circumflex, caret
95	_	Underscore, underline
96	`	grave accent
97	a	Lowercase a

(continued on next page)

Table C-1: DATATRIEVE Sort Order (Cont.)

Decimal Value	Character	Description
98	b	Lowercase b
99	c	Lowercase c
100	d	Lowercase d
101	e	Lowercase e
102	f	Lowercase f
103	g	Lowercase g
104	h	Lowercase h
105	i	Lowercase i
106	j	Lowercase j
107	k	Lowercase k
108	l	Lowercase l
109	m	Lowercase m
110	n	Lowercase n
111	o	Lowercase o
112	p	Lowercase p
113	q	Lowercase q
114	r	Lowercase r
115	s	Lowercase s
116	t	Lowercase t
117	u	Lowercase u
118	v	Lowercase v
119	w	Lowercase w
120	x	Lowercase x
121	y	Lowercase y
122	z	Lowercase z
123	{	Left brace
124		Vertical line
125	}	Right brace
126	~	Tilde

Edit String Characters **D**

The following table lists and describes all the edit string characters you can use in the `EDIT_STRING` clause of a field definition or in the `USING` clause of the `PRINT` statement.

The table heading “Character Type” indicates what type of field you can edit with the character. Do not use editing characters designated only alphabetic or alphanumeric on numeric fields (or vice versa). If you do, you can get unexpected results. Remember that field type is determined by the `PIC` or `USAGE` clause, not the value in the field. A field defined as `PIC X(10)` might contain only numbers, for example, but you should use only alphanumeric editing characters to format the way you want to display those numbers.

Table D-1: Edit String Characters

Character Type	Edit String Character	Description
Alphabetic Replacement	A	Each A is replaced by an alphabetic character from the field's content. An asterisk is placed in the position of each digit or nonalphabetic character in the field's content.
Alphanumeric Replacement	X	Each X is replaced by one character from the field's content.

(continued on next page)

Table D-1: Edit String Characters (Cont.)

Character Type	Edit String Character	Description
Numeric Replacement	T	Indicates text. Each T reserves a column on a line for the associated print list element. For example, PRINT "1234567890" USING T(5) displays: 12345 in the first five columns of one line and 67890 in the first five columns of the next line. Edit strings containing a T cannot contain other characters.
	A	Each A is replaced by an alphabetic character from the field's content. An asterisk is placed in the position of each digit or nonalphabetic character in the field's content.
	9	Each 9 is replaced by one digit from the field's content. Nondigit characters are ignored, and the digits are right justified in the output and the leading character positions (if any) are filled with zeros.
	Z	If a Z matches a leading zero in the field's content, it is replaced by a space. If not, Z is replaced by a digit from the field's content.
	* (asterisk)	If an asterisk (*) matches a leading zero in the field's content, an asterisk is placed in that character position. If not, it is replaced by a digit from the field's content.
Alphanumeric Insertion	. (period)	A period specifies the character position of the decimal point.
	+ (plus)	A plus is inserted in that character position unless more than one plus sign is specified.
	- (hyphen)	A hyphen is inserted in that character position.
	. (period)	A period is inserted in that character position.
Numeric Insertion	, (comma)	A comma is inserted in that character position.
	+ (plus)	If only one plus sign is specified, it is replaced by either a plus sign, if the field's content is positive, or a minus sign if it is negative.

(continued on next page)

Table D-1: Edit String Characters (Cont.)

Character Type	Edit String Character	Description
Alphanumeric and Numeric Insertion	– (minus)	If only one minus sign is specified, it is replaced by either a blank, if the field's content is positive, or a minus sign if it is negative.
	. (decimal)	A decimal point is inserted in that character position. Put only one decimal point in a numeric edit string.
	, (comma)	If all the digits to the left of the comma are suppressed zeros, the comma is replaced by a blank. If not, a comma is inserted in that character position.
	CR	If the field's content is negative, the letters CR are inserted. If the field's content is positive, CR is replaced by 2 blanks. Put only one CR in an edit string, either at the far right or the far left.
	DB	If the field's content is negative, the letters DB are inserted. If the field's content is positive, DB is replaced by 2 blanks. Put only one DB in an edit string, either at the far right or the far left.
	(()) (parentheses)	If the field's content is negative, single left and right parentheses are inserted before and after the field value.
	B	A space is inserted in that character position.
	0 (zero)	A zero is placed in that character position.
	\$ (dollar sign)	If only one dollar sign is specified, it is printed in that character position.
	% (percent)	A percent sign is inserted in that character position.
/ (slash)	A slash is inserted in that character position.	
"literal"	The character string literal enclosed in single or double quotation marks is inserted at that position. The outer quotation marks are not inserted in the output.	

(continued on next page)

Table D-1: Edit String Characters (Cont.)

Character Type	Edit String Character	Description
Numeric Floating Insertion	\$ (dollar sign)	If more than one dollar sign is specified to the left of the other edit string characters, leading zeros are suppressed and one dollar sign is displayed to the immediate left of the leftmost digit.
	+ (plus)	If more than one plus sign is specified to the left of the other edit string characters, any leading zeros are suppressed, and the sign of the field's value (plus or minus) is displayed to the immediate left of the leftmost character position determined by the other edit string characters.
	- (minus)	If more than one minus sign is specified to the left of the other edit string characters, any leading zeros that the minus sign matches are suppressed. If the value of the field is negative, a minus sign is displayed to the immediate left of the leftmost character position determined by the other edit string characters.
Floating-Point Edit String	E	The E divides the edit string into two parts for floating-point or scientific notation. The first part is the mantissa edit string and the second part is the exponent edit string.
Missing Value Edit String	?	If the field has a MISSING VALUE clause, the question mark separates two edit strings. If the field value is not the missing value, the first edit string controls the output of the field. If the field value is the missing value, the second edit string controls the output of the field.
Date Replacement	D	Each D is replaced by the corresponding digit of the day of the month, starting with the first letter. Put no more than two Ds in a date edit string; the use of DD is recommended.
	M	Each M is replaced by the corresponding letter of the name of the month. An edit string of M(9) prints the entire name of the month.

(continued on next page)

Table D-1: Edit String Characters (Cont.)

Character Type	Edit String Character	Description
	N	Each N is replaced by a digit of the number of the month. Put no more than two Ns in a date edit string; the use of NN is recommended.
	Y	Each Y is replaced by the corresponding digit of the numeric year. Put no more than four Ys in a date edit string; the use of YY or YYYY is recommended.
	J	Each J is replaced by the corresponding digit of the Julian date. Put no more than three Js in a date edit string; the use of JJJ is recommended.
	W	Each W is replaced by the corresponding letter from the name of the day of the week, starting with the first letter. An edit string of W(9) prints the entire day. Put no more than 9 Ws in a date edit string.
	B	Each B is replaced by a space in that character position.
	/ (slash)	A slash is inserted in that character position.
	- (hyphen)	A hyphen is inserted in that character position.
	. (period)	A period is inserted in that character position.



Index

In this index, a page number followed by a "t" indicates a table reference.

! (exclamation point)

See Comments

\$ (dollar sign)

See Dollar sign (\$)

% (percent sign)

See Percent sign (%)

* (asterisk)

See Asterisk (*)

- (hyphen)

See Hyphen (-)

: (colon)

See Colon (:)

; (semicolon)

See Semicolon (;)

? (question mark)

See Question mark (?)

A

A (alphabetic)

edit string character, D-1t

A (alphabetic) picture string character,
9-10t

Abbreviating

DATATRIEVE keywords, 5-2

DCL command strings, 2-28

DCL commands, 2-6

ABORT statement, 17-15

Access control list, 3-7, 7-16

editing, 7-21

Access modes, 13-4t

EXTEND, 13-4t

MODIFY, 13-4t

READ, 13-4t

WRITE, 13-4t

Access options, 13-4t

EXCLUSIVE, 13-4t

PROTECTED, 13-4t

SHARED, 13-4t

Access privileges, 7-16

displaying, 7-18

in multiuser environments, 13-4

needed to use

domains, 13-4

procedures, 7-19

record definitions, 10-3

tables, 12-5

views, 15-19

requirements, 7-19t

Accessing data, 15-20

Account, 2-2

ACL

See Access control list

ADT
 See Application Design Tool
AFTER relational operator, 18-11t
Alias, 13-3
 using to restructure domains, 11-8
ALL keyword
 in PRINT statement, 19-11
ALLOCATION clause, 11-6
Alphabetic fields, 9-10
Alternate keys
 allowing
 changes to, 11-2
 duplicate values in, 11-2
 characteristics of, 11-2
 choosing, 11-4
AND Boolean operator, 18-14
ANY relational operator, 18-11t
 and list fields, 18-14
APPEND command (DCL), 2-17
Application Design Tool (ADT), 1-17
 to 1-19
Arithmetic operations, 18-7t
 order of evaluation for, 18-8
Arithmetic value expressions, 18-7
 rules for evaluating, 18-8
ASCII values for characters, C-1
ASSIGN command (DCL), 2-24
 /USER_MODE qualifier, 2-26
Assigning default editor, 3-2
Asterisk (*)
 arithmetic operator (multiplication),
 18-7
 as wildcard character (DCL), 2-12
 edit string character, D-1t
 in prompting value expressions,
 18-6
AT BOTTOM statement
 REPORT option, 20-2
 to summarize control group data,
 20-13
AT TOP statement
 to create title page, 20-16
 to specify control group header,
 20-13
Auditing information

 including time in, 16-15
 writing to a file, 16-8
AVERAGE statistical operator, 18-17

B

B (blank)
 edit string character, D-1t
BEFORE relational operator, 18-11t
BEGIN-END statement, 17-8
 declaring variables in, 18-5
BETWEEN relational operator,
 18-11t
Blank lines
 adding to displays, 19-11
 including in reports, 20-14
Boolean expressions, 18-1
 compound, 18-15
 contents of, 18-10
 evaluation of, 18-15
 uses for, 18-11
Boolean operators, 18-14
Braces in syntax diagrams, 4-2t, 4-4
Brackets in syntax diagrams, 4-2t, 4-4
BT
 See BETWEEN relational operator
BYTE data type, 9-13t

C

Call Interface, reasons for using, 1-31
Case conversion of
 CDD names, 7-4
 field values, 7-4
Case-sensitivity and character values,
 18-13
CDD
 See Common Data Dictionary
CDD\$DEFAULT logical name, 1-6
CDD\$TOP dictionary directory, 7-3
Character strings
 searching for in files, 2-17
 stored in fields, 9-10
CHOICE statement, 17-11
CHOICE value expression, 18-9
CLOSE command, 17-18

- COL print list element, 17-3, 19-11
 - in reports, 20-14
- Collections, 14-1
 - advantages of, 15-20
 - changing access mode for, 14-16
 - choosing target record from, 14-7
 - compound statements with, 15-4
 - CURRENT, 14-6
 - performing statistical operations on, 18-20
 - disadvantages of, 14-16, 15-20
 - displaying
 - information about, 14-6
 - names of, 14-6
 - do not use keys, 15-20
 - dropping records from, 14-15
 - effect on performance, 14-17
 - erasing records from, 16-5
 - how they work, 15-1
 - modifying records in
 - to assign different values, 16-13
 - to assign one value per field, 16-11 to 16-13
 - naming, 14-6
 - reducing number of fields in, 14-10
 - removing from workspace, 14-16
- Colon (:)

 - following logical name, 2-24
 - using to execute procedures, 17-1
 - from command files, 17-14

- Column headers
 - in SUM statement, 18-21
 - PRINT statement, 19-13
 - QUERY_HEADER clause, 9-7
 - to optimize line space, 19-4
- Comma (,)

 - edit string character, D-1t
 - following an edit string, 4-5

- Command files, 2-27
 - converting to procedures, 17-17
 - correcting problems in
 - using interactive entry, 17-18
 - using log files, 17-18
 - using SET VERIFY, 17-16
 - created with EXTRACT, 7-13
- DATATRIEVE startup, 5-2
- disadvantages of, 17-18
- LOGIN.COM, 2-28
 - nesting, 2-28
- Command string (DCL), 2-1
- Commands
 - continuing over lines, 5-4 to 5-6
 - continuing with hyphen, 5-5
 - ending, 4-5, 5-6, 7-9
 - privileges needed to use, 7-19
- Comments, in procedures, 17-2
- Common Data Dictionary
 - access control, 7-16
 - creating directories in, 7-14
 - deleting
 - definitions in, 7-12
 - directories in, 7-15
 - full path names in, 7-5
 - functions of, 7-1
 - organization of, 7-2
 - relative path names in, 7-7
 - rules for names in, 7-4
 - setting location in, 7-9
 - structure of, 1-15
 - used in examples, 7-3
- COMP data type, 9-13t
- COMP-1 data type, 9-13t
- COMP-2 data type, 9-13t
- COMP-3 data type, 9-13t
- COMP-5 data type, 9-13t
- Comparing
 - character string values, 18-13
 - file contents, 2-17
- Compound statements
 - See* Statements, compound
- COMPUTED BY clause, 9-15
- CON > prompt, 1-12, 5-4
- Concatenated expressions
 - to format names, 17-2
- Concatenation characters, 19-7
- Conditional value expressions, 18-8
 - CHOICE, 18-9
 - IF-THEN-ELSE, 18-8
- CONTAINING relational operator, 18-11t

- Context errors, 1-14
 - when accessing list fields, 15-11
 - Continuation character (-), 5-5, 19-3
 - CONTINUE command (DCL), 2-8
 - Continuing
 - commands, 5-4 to 5-6
 - literals, 19-3
 - statements, 5-4 to 5-6
 - Control groups, 20-12
 - CONTROL privilege, 7-17
 - COPY command (DCL), 2-15
 - Copying
 - definitions to files, 7-13
 - files, 2-15
 - Correcting mistakes
 - with EDIT command, 1-14
 - COUNT statistical operator, 18-17
 - CR (credit)
 - edit string character, D-1t
 - CREATE/DIRECTORY command (DCL), 2-20
 - Creating
 - DATATRIVEE procedures, 17-1
 - domain definitions, 10-1
 - files, 11-1
 - indexed, 11-2
 - sequential, 11-5
 - record definitions, 9-1
 - tables
 - dictionary, 12-2 to 12-3
 - domain, 12-3 to 12-4
 - variables, 18-4
 - CROSS clause, 14-4
 - cannot erase records created by, 16-5
 - FIND statement RSE, 14-14
 - in view definition, 15-15
 - looping errors in, 15-10 to 15-11
 - modifying records created by, 16-11
 - CTRL/A, 2-7
 - CTRL/B, 2-7
 - CTRL/C
 - canceling DCL command, 2-7
 - effect on procedure execution, 17-14
 - exiting Report Writer, 20-19
 - CTRL/E
 - at DCL level, 2-7
 - expanding LSE placeholders and tokens, 3-15
 - CTRL/H, 2-7
 - CTRL/Q, 2-7
 - CTRL/S, 2-7
 - CTRL/U, 2-7
 - CTRL/Y
 - canceling DCL commands, 2-7
 - exiting DATATRIVEE with, 5-3
 - CTRL/Z
 - canceling DCL command, 2-7
 - effect on procedure execution, 17-15
 - exiting DATATRIVEE, 5-3
 - stopping a store operation, 16-2
 - using in editors, 3-16
 - EDT, 3-16
 - LSE, 3-17
 - VAXTPU, 3-17
- ## D
- D (day number)
 - edit string character, D-1t
 - DAT file type, 2-11t
 - Data
 - summary of access options, 15-20 to 15-21
 - Data files
 - See also* Indexed files
 - See also* Sequential files
 - accessing using domain definitions, 10-1
 - adjusting I/O parameters for, 11-7
 - changing organization of, 11-7
 - consolidating storage for, 11-7
 - creating with
 - DEFINE FILE command, 11-1
 - MAX option, 11-6
 - RMS utilities, 11-7
 - SUPERSEDE option, 11-6
 - location of, 1-2

- maintaining, 1-30
 - effect of key structure on, 11-4
 - with RMS utilities, 11-7
 - preallocating disk space for, 11-6
 - specifying in domain definitions, 10-3
- Data types
 - ASCII string, 9-13t
 - assigning with PIC clause, 9-9
 - date, 9-13t
 - fixed decimal, 9-13t
 - floating point, 9-13t
- Database design
 - access restrictions in, 8-3
 - data entry requirements, 8-2
 - deciding on fields in, 8-4 to 8-6
 - distributed or local access, 8-2
 - grouping fields in, 8-6 to 8-9
 - key fields in, 8-7
 - maintenance requirements, 8-3
 - report requirements, 8-2
 - system requirements, 8-2
- Databases accessed by
 - DATATRIEVE
 - DBMS databases, 1-2
 - Rdb/ELN databases, 1-2
 - Rdb/VMS databases, 1-2
 - Relational databases, 1-2
 - VIDA databases, 1-2
- DATATRIEVE
 - differences from other languages, 1-29 to 1-31
 - storage capabilities, 1-29
- DATATRIEVE editors
 - See also* EDIT command
 - See also* EDT editor
 - See also* LSE
 - See also* VAXTPU editor
 - assigning default editor, 3-2
- Date fields
 - controlling conversion of numeric strings for, 18-23
 - default display of, 9-15
 - defining, 9-15
 - displaying time stored in, 18-23
 - displaying values in, 18-24
 - searching for values in, 18-26
 - to find day of week, 18-28
 - size of, 9-15
 - storing
 - date values in, 18-23
 - time values in, 18-23
 - subtracting, 17-12, 18-26
- DB (debit)
 - edit string character, D-1t
- DBMS databases, 1-2, 1-30
- DCL
 - See* DIGITAL Command Language (DCL)
- DEASSIGN command (DCL), 2-26
- Decimal point
 - specifying position of
 - SCALE clause, 9-14
- Decimal point, specifying position of
 - P character, 9-11
 - SCALE clause, 9-11
 - V character, 9-11
- DECLARE statement, 18-4
- DECLARE SYNONYM command, 5-2
- Defaults
 - access mode, 13-3
 - access option, 13-3
 - for column headers, 9-7
 - for data types, 9-12
 - for displaying dates, 9-15
 - for field value displays, 9-20
 - for index keys, 11-2
 - for order of records in indexed files, 11-3
 - location
 - of data files, 10-3
 - of record definitions, 10-3
 - operating system, 2-6
- DEFINE DICTIONARY command, 1-16, 7-14
 - access privilege requirements, 7-21
- DEFINE DOMAIN command, 10-1
 - access privilege requirements, 7-21
 - to create view definition, 15-14

- DEFINE FILE command, 11-1
 - access privilege requirements, 7-21
 - ALLOCATION option, 11-6
 - ending, 11-2
 - KEY clause options, 11-2
 - MAX option, 11-6
 - SUPERSEDE option, 11-6
- DEFINE PORT command, access
 - privilege requirements, 7-21
- DEFINE PROCEDURE command,
 - 7-21, 17-1
- DEFINE RECORD command, access
 - privilege requirements, 7-21
- DEFINE TABLE command
 - access privilege requirements, 7-21
 - EDIT_STRING clause, 12-2, 12-4
 - ELSE clause, 12-2, 12-4
 - END_TABLE requirement, 12-2, 12-4
 - for dictionary tables, 12-2 to 12-3
 - for domain tables, 12-3 to 12-4
 - QUERY_HEADER clause, 12-2, 12-4
- DELETE command
 - access privilege requirements, 7-21
 - DATATRIEVE, 7-12
 - DCL, 2-13
 - for deleting subdirectory, 2-23
 - /SYMBOL qualifier, 2-26
 - DMU, 7-15
- DELETE key, 2-6
- DELETEP command, access privilege
 - requirements, 7-21
- Deleting
 - definitions
 - all earlier versions of, 7-12
 - files, 2-13
 - records
 - See Erasing records
- Devices
 - in file specifications, 2-11
 - writing PRINT statement output
 - to, 19-13
- DFN > prompt, 1-25
- Dictionary path names, 7-7
- DIFFERENCES command (DCL),
 - 2-17
- DIGITAL Command Language (DCL),
 - 2-1
 - abbreviating commands in, 2-6
 - CONTINUE command, 2-8
 - defaults, 2-6
 - delimiters, 2-5
 - entering commands in, 2-4
 - messages, 2-8
 - parameters, 2-4
 - qualifiers, 2-5
- Directories
 - CDD, 1-6
 - assigning CDD\$DEFAULT to, 1-6
 - creating, 1-16, 7-14
 - deleting, 7-15
 - displaying contents of, 1-9
 - displaying default, 1-8
 - VMS, 1-2
 - accessing other, 2-19
 - changing location among, 2-20
 - creating, 1-4, 2-20
 - default, 1-5
 - deleting, 2-23
 - DIR files, 2-18
 - displaying contents of, 1-3, 2-15
 - displaying protection for, 2-22
 - entering names of, 1-5
 - in file specifications, 2-11
 - main (login), 1-3
 - master file of, 2-18
 - protection for, 2-21
- DIRECTORY command (DCL), 2-15
 - /OWNER option, 2-21
- Displaying
 - access privileges, 7-18
 - CDD objects, 7-10t
 - command files as they execute, 17-16
 - data in views, 15-15
 - fields in readied domains, 14-5
 - files, 2-14
 - information

- about a collection, 14-6
- stored in dictionary, 7-9
- items at specific line positions, 17-3
- names of
 - CDD directories, 7-10t
 - collections, 14-6
 - DATATRIEVE procedures, 7-10t
 - domains, 10-2
 - loaded tables, 12-7
 - readied domains, 13-1
 - records, 7-10t
 - tables, 12-3
 - variables, 7-10t
- OCCURS field items, 19-11
- procedures as they execute, 17-16
- protection for
 - files, 2-22
 - VMS directories, 2-22
- records, 1-21
 - longer than 80 characters, 19-1
- translation of
 - logical name, 2-26
 - logical symbol, 2-26
- view definitions, 15-15
- VMS directory contents, 2-15
- Displays, stopping
 - with CTRL/C, 1-12
 - with NO SCROLL key, 1-11
- Distributed data, 10-1
 - accessing, 10-4
- Division arithmetic operator (/), 18-7
- Dollar sign (\$)
 - edit string character, D-1t
 - prompt, 1-2, 2-3
- Domains
 - accessing
 - data from two or more with view, 15-14
 - data in two or more with CROSS clause, 15-10
 - subsets of fields in, 15-14
 - changing structure of, 11-7
 - creating, 10-1 to 10-2
 - displaying readied, 13-1
 - purpose of, 10-1
 - rules for naming, 9-7, 10-2
- DOUBLE data type, 9-13t
- DROP statement, 14-15
- DTR\$DATE_INPUT logical name, 18-23
- DTR\$LIB dictionary directory, 7-3
- DTR\$STARTUP logical name, 5-2
 - assigning command file to, 5-2
- DTR\$USERS dictionary directory, 7-3
- DTR32 logical symbol
 - including in LOGIN.COM, 1-6
- DTR> prompt, 1-8
- DTR_EXTEND/EXECUTE privilege, 7-17
- DTR_MODIFY privilege, 7-17
- DTR_READ privilege, 7-17
- DTR_WRITE privilege, 7-17

E

- E (floating point)
 - edit string character, D-1t
- EDIT command, 3-5
 - access privilege requirements, 7-21
 - correcting mistakes with, 1-14
 - RECOVER argument, 3-8
 - specifying CDD path name, 3-7
 - specifying object types, 3-7
- Edit string characters, D-1t
- Edit strings
 - alphabetic replacement, D-1t
 - alphanumeric insertion, D-1t
 - alphanumeric replacement, D-1t
 - for date fields, 18-24
 - for statistical results, 18-18
 - in FORMAT value expressions, 18-10
 - in SUM statement, 18-21
 - numeric floating insertion, D-1t
 - numeric insertion, D-1t
 - numeric replacement, D-1t
 - problems when too small, 19-6
 - specifying in PRINT statement, 19-13

- that display time, 18-23
- using to optimize line space, 19-6
- EDIT_STRING clause, 9-20
- Editing
 - See also* DATATRIEVE editors
 - See also* EDT editor
 - See also* LSE
 - See also* VAXTPU editor
 - access control lists, 7-21
 - buffers, 1-7
 - in EDT keypad mode, 3-10
 - in EDT line mode, 3-10
 - record definitions, 9-24
 - table definitions, 12-7
- Editors
 - See* DATATRIEVE editors
 - See* Editing
- EDT editor, 3-1
 - exiting, 3-16
 - invoking from DATATRIEVE, 3-9
 - journal files, 3-8
 - quitting, 3-16
- EDT file type, 2-11t
- EDT Keypad Emulator, 3-11
- Elementary fields, 1-17
- Ellipsis in syntax diagrams, 4-2t
- Ending
 - access to data, 1-14
 - statements with semicolon, 5-6
- Ending editing sessions
 - EXIT command, 3-16
 - QUIT command, 3-16
- EQUAL relational operator, 18-11t
- Equal sign (=)
 - as relational operator, 18-11t
 - in symbol definition, 2-28
- Erasing
 - definitions
 - See* DELETE command
 - dictionary directories
 - See* DELETE command, DMU
 - records, 16-5
 - from a collection, 16-5 to 16-7
 - in a FOR statement RSE, 16-7 to 16-10

- Error conditions
 - in procedures
 - with SET ABORT, 17-14
 - specifying, 17-15
- Error messages
 - getting help for, 6-2
 - in response to DCL commands, 2-8
- EXCLUSIVE access option, 13-4t
- EXE file type, 2-11t
- EXECUTE (:) command, 17-1
- Executing
 - command files, 2-27
 - DATATRIEVE procedures, 17-1
- EXIT command in editing, 3-16
- Exiting
 - DATATRIEVE, 1-15, 5-3
 - when not at DTR> prompt, 5-3
 - with CTRL/Y, 5-3
 - DATATRIEVE online help, 6-2
 - EDT editor, 1-14, 3-16
 - LSE, 3-17
 - Report Writer
 - with CTRL/C, 20-19
 - with END_REPORT, 20-19
 - VAXTPU editor, 3-17
- EXTEND access mode, 7-21, 13-4t
- EXTEND privilege, 7-17
- Extensible VAX Editor (EVE), 3-11
- EXTRACT command, 7-13, 17-17
 - access privilege requirements, 7-21

F

- Field names
 - abbreviating, 9-7
 - as value expressions, 18-4
- Field values
 - character string
 - sort order of, C-1
 - performing arithmetic operations
 - with, 18-7
- Fields
 - record
 - alphabetic, 9-10
 - changing values of, 1-22

- DATATRIEVE recognition of,
 - 9-4
 - display headers for, 9-7
 - duplicate, 9-6
 - elementary, 9-4
 - formatting values of, 9-20
 - group, 9-4
 - numeric, 9-11
 - redefining other fields, 9-16
 - relationships among, 9-3
 - restricting characters allowed in,
 - 9-9
 - size of, 9-9
 - specifying decimal point in, 9-11
 - specifying size of, 9-13t
 - storing dates, 9-15
 - top-level, 9-4
 - type of data in, 9-9
 - using tables to validate, 12-8
- variable
 - computed, 9-16
 - defining, 18-4 to 18-5
- File specifications
 - errors in, 10-4
 - format for, 2-10
 - in domain definitions, 10-3
 - including logical names in, 2-24
- File types
 - See also* DAT file type
 - See also* EDT file type
 - See also* EXE file type
 - See also* Journal files
 - See also* LIS file type
 - See also* MAI file type
 - See also* OBJ file type
 - defaults, 2-11t
 - restrictions, 2-11t
- Files
 - See also* Data files
 - appending, 2-17
 - changing default protection for,
 - 2-22
 - creating with
 - CREATE command (DCL), 2-10
 - deleting, 2-13
 - displaying, 2-14
 - displaying protection for, 2-22
 - journal, 3-8
 - listing differences between, 2-17
 - moving
 - with COPY command, 2-15
 - with RENAME command, 2-16
 - naming, 2-11
 - printing, 2-14
 - protection codes for, 2-21
 - purging, 2-13
 - recovering, 3-8
 - renaming, 2-16
 - setting protection for, 2-22
 - writing PRINT statement output
 - to, 19-13
- FILLER field name
 - does not mask sensitive data, 9-9
 - uses for, 9-8
- FIND statement
 - advantages of using, 15-20
 - disadvantages of using, 15-20
 - displays no message in procedure,
 - 17-13
- FINISH command, 13-1
 - effect of, 13-6
- FIRST clause of RSE, 14-4
- Fixed decimal data types, 9-13t
- Fixed occurrence lists, 9-18
- Floating point data types, 9-13t
- FN\$DATE function, 18-23
- FN\$WIDTH function, 19-2
- FOR statement
 - erasing records specified in a, 16-7
 - to 16-10
 - looping errors in a, 15-10
 - results, 17-7
 - used in a modify operation, 16-13 to
 - 16-15
- Foreign command lines, 5-1
- FORMAT value expression, 18-10
 - to control alignment of numeric values, 18-10
 - to search for day of the week, 18-28

using with MAX value expression,
16-3
FORWARD privilege, 7-17
Fractions, ensuring display of, 9-20
Functions, A-5t

G

G_FLOATING data type, 9-13t
GE
 See GREATER_EQUAL relational
 operator
Global variables, 18-5
GLOBAL_DELETE privilege, 7-17
GREATER_EQUAL relational opera-
tor, 18-11t
GREATER_THAN relational opera-
tor, 18-11t
Group fields, 1-17
 are always alphanumeric, 9-16
 as index keys, 11-4
 restrictions on, 11-5
GT
 See GREATER_THAN relational
 operator
Guide Mode
 entering, 6-3
 using advanced, 6-4

H

H_FLOATING data type, 9-13t
Headers
 See Column headers
Help
 DATATRIEVE online
 accessing, 6-1
 controlling displays in, 6-2
 exiting, 6-2
 for error messages, 6-2
 topic levels for, 6-2
 VMS online, 2-4
History lists, 7-23
HISTORY privilege, 7-17
HOLD SCREEN key, 2-7
Hyphen (-)

cannot end a comment line, 17-2
continuation character, 5-5, 19-3
edit string character, D-1t
in path names, 7-7, 7-9
to suppress column headers, 19-5

I

IF-THEN-ELSE statement, 17-10
IF-THEN-ELSE value expression,
18-8
Indentation
 in compound statements, 17-7
 in record definitions, 9-5
Index keys in database design, 8-7
Indexed files
 advantages of, 11-2
 and collections, 15-1
 changing keys for, 11-7
 for transaction file processing, 11-3
 specifying
 alternate keys for, 11-4
 group field keys for, 11-4
 primary keys for, 11-3
INTEGER data type, 9-13t
Invocation command lines, 5-1

J

J (Julian date)
 edit string character, D-1t
JOU file type
 See Journal files
Journal files
 JOU file type, 2-11t, 3-8
 TJL file type, 3-8
Julian date
 See J (Julian date) edit string
 character

K

Keywords, A-1t
 creating synonyms for, 5-2
 effect on user-defined names, 4-4
 optional, 4-4

required, 4-4

L

Language-Sensitive Editor

See LSE

LE

See LESS_EQUAL relational operator

LESS_EQUAL relational operator, 18-11t

LESS_THAN relational operator, 18-11t

Level numbers, 9-3

for redefining fields, 9-16

leading zeros in, 9-5

range of values for, 9-5

suggested intervals between, 9-5

Line recall, 3-3

LIS file type, 2-11t

List fields

cannot be index keys, 11-5

creating loops to process, 15-11 to 15-13

defining, 9-17

disadvantages of, 9-18

fixed occurrence, 9-18

group field names for, 9-19

using ANY operator to search, 18-14

variable occurrence, 9-18

LIST statement, 19-2

Literals, 18-3

character string, 18-3

case of letters in, 18-3

searching for in files, 2-17

continuing to next line, 19-3

longer than 253 characters, 19-8

numeric, 18-3

Local variables, 18-5

LOCAL_DELETE privilege, 7-17

Log files, creating, 17-18

Logging in, 2-2

Logging out, 2-9

Logical names

advantages of using, 2-23

CDD\$DEFAULT, 1-6

default, 2-24

deleting, 2-26

DTR\$DATE_INPUT, 18-23

DTR\$STARTUP, 5-2

SYSDISK, 2-25

SY\$ERROR, 2-25

SY\$INPUT, 2-25

SY\$OUTPUT, 2-25

system tables of, 2-24

group, 2-24

job, 2-24

process, 2-24

system, 2-24

temporary, 2-26

translation of, 2-25

LOGIN.COM file, 2-28

assigning CDD\$DEFAULT, 1-6

including DTR32 symbol, 1-6

LONG data type, 9-13t

"Looking for" messages, turning on and off, 5-5

Loops

avoiding errors when creating, 17-13

conditions for executing, 17-9

errors when creating

CROSS clause, 15-10 to 15-11

FOR statement, 15-10

infinite, 17-13

processing

lists with, 15-11 to 15-13

multiple records with, 15-9

Lowercase words in syntax diagrams,

4-2t, 4-4

LSE, 3-13

DATATRIEVE templates, 3-13

exiting, 3-17

invoking from DATATRIEVE, 3-14

journal files, 3-8

placeholders, 3-13

quitting, 3-17

tokens, 3-13

LT

See LESS_THAN relational operator

M

M (month letter)

edit string character, D-1t

MAI file type, 2-11t

Master file directories (MFD), 2-18

MAX (maximum value) statistical operator, 18-17

MAX option of DEFINE FILE command, 11-6

MFD

See Master file directories (MFD)

MIN (minimum value) statistical operator, 18-17

Minus sign (-)

arithmetic operator, 18-7

edit string character, D-1t

interpreted as underscore, 18-7

MISSING relational operator, 18-11t

MISSING VALUE clause and statistical operations, 18-17

MODIFY access mode, 7-21, 13-4t

Modifying records, 1-22

collection

to assign different values, 16-13

to assign one value per field, 16-11 to 16-13

created by CROSS clause, 16-11

FOR statement loop, 16-13 to 16-15

in views, 16-11

MODIFY statement RSE, 16-13

reading domains when, 16-10

using a transaction file, 16-13 to 16-15

Multiplication arithmetic operator (*), 18-7

N

N (month number)

edit string character, D-1t

Names

conventions for

record and domain, 9-7

top-level field, 9-7

duplicate field, 9-6

of

domains, 10-2

procedures, 17-1

record definitions, 9-5

tables, 12-1

qualifying field, 9-6

record

related to top-level field, 9-6

used out of context, 1-14

NE

See NOT_EQUAL relational operator

NEW_PAGE print list element, 19-11

in reports, 20-14

NEWUSER program, 1-6

error messages from, 1-8

NO SCROLL key, 2-7

Nodes in file specifications, 2-10

NOT Boolean operator, 18-14

NOT_EQUAL relational operator, 18-11t

NOW value expression, 18-23

used in a procedure, 16-15

Numeric fields

allowing negative values in, 9-12

calculations with, 9-12

defining

PIC clause, 9-11

the easy way, 9-12

USAGE clause, 9-13

maximum number of digits in, 9-12

specifying size of, 9-13

that need edit strings, 9-20

9 (numeric)

edit string character, D-1t

picture string character, 9-10t

O

OBJ file type, 2-11t

Object types, specifying with EDIT command, 3-7

OCCURS clause, 9-17
 disadvantages of, 1-30
 variable definition restriction, 18-4
 ON clause of PRINT statement, 19-13
 ON statement, 19-13
 used to print auditing information,
 16-8
 Online assistance
See Help
 OPEN command, 17-18
 Operating system, 2-1
 Operators
 arithmetic, 18-7
 Boolean, 18-14
 relational, 18-11
 Optimizing
See also Performance
 data files, 1-30
 OR Boolean operator, 18-14
 Ordering records
 in RSEs, 14-4
 SORT statement, 14-12
 OVER clause of RSE, 14-4

P

P (decimal scaling) picture string character, 9-10t
 PACKED data type, 9-13t
 Page breaks, specifying in reports,
 20-14
 Parameter (DCL), 2-1
 Parentheses
 edit string characters, D-1t
 to group Boolean expressions,
 18-15
 to order arithmetic operations, 18-8
 PASS_THRU privilege, 7-17
 Password (login), 2-2
 Path names, 1-15
 contents of
 full, 7-5
 relative, 7-7
 displaying privileges, 7-18
 full

of domain in domain definitions,
 10-3
 of records in domain definitions,
 10-3
 version numbers in, 7-5
 Percent sign (%)
 edit string character, D-1t
 Performance
 as affected by
 data file maintenance, 11-7
 duplicate alternate keys, 11-4
 duplicate primary keys, 11-3
 using domains to improve, 15-1
 when using
 collections, 14-17, 15-20
 domains directly, 15-20
 tables, 12-2
 Period (.)
 edit string character, D-1t
 Period (.) in path names, 7-5
 PICTURE clause, 9-9
 Picture string characters, 9-10t
 A (alphabetic), 9-10t
 9 (numeric), 9-10t
 P (decimal scaling), 9-10t
 S (sign), 9-10t
 V (decimal point), 9-10t
 X (alphanumeric), 9-10t
 Plus sign (+)
 edit string character, D-1t
 Plus sign (+) arithmetic operator,
 18-7
 Precision in arithmetic operations
 with numeric USAGE fields, 9-13
 with PIC fields, 9-12
 Primary keys
 allowing duplicate values in, 11-2
 cannot allow changes to, 11-2
 characteristics of, 11-2
 choosing, 11-3
 for transaction file processing, 11-3
 PRINT command (DCL), 2-14
 Print list, 19-9
 Print list elements, 19-11t
 Print list modifiers, 19-13t

Print list with OCCURS field, 19-11

PRINT statement

- how to enter a, 19-9
- in a report specification, 20-2
- output to file or device, 19-13

Procedures

- access privileges for invoking, 7-21
- advantages of, 1-31
- cannot invoke themselves, 17-13
- comment lines in, 17-2
- controlling error conditions in, 17-14
- converting to command files, 17-16
- correcting problems in, 17-16
 - using interactive entry, 17-18
 - using log files, 17-18
 - using SET VERIFY, 17-16
- creating, 1-27, 17-1
 - reports with, 20-3
- editing, 17-4
- ending definitions of, 17-4
- erasing records with, 16-7 to 16-10
- error conditions in
 - specifying, 17-15
- executing, 1-27, 17-1
- invoking from command files, 17-14
- invoking in loops, 17-13
- modifying records with, 16-13 to 16-15
- naming, 17-1
- not compiled when stored, 1-31
- storing records with, 16-2
- using collections in, 14-16

Prompting for

- input
 - in a modify operation, 16-14
 - when storing records, 16-3
- output file or device, 19-14

Prompting value expressions, 18-6

- using one asterisk (*), 18-6
- using two asterisks (**), 18-6

Prompts

- CON >, 1-12, 5-4
- DFN >, 1-25
- dollar sign (\$), 1-2
- DTR >, 1-8
- RW >, 20-2

PROTECTED access option, 13-4t

Protecting

- CDD directories, 7-16
 - from other users, 7-21
 - in multiuser applications, 7-23
- definitions, 7-16
 - from other users, 7-21
 - in multiuser applications, 7-23
- files
 - changing default for, 2-22
 - codes for, 2-21
 - SET PROTECTION command, 2-22
- VMS directories, 2-21

Punctuation

- in syntax diagrams, 4-2t
- to end DATATRIEVE statements, 4-5

PURGE command, 7-12

- DCL, 2-13

Purging

- DCL files, 2-13
- definitions, 7-13

Q

QUAD data type, 9-13t

Qualified field names, 9-6

Qualifier (DCL), 2-1

Query names

- in SHOW FIELDS display, 1-12
- uses for, 9-7

QUERY_HEADER clause, 9-7

- using to reduce size of column headers, 19-5

QUERY_NAME clause, 9-7

Question mark (?)

- entering for online help, 6-1
- missing value edit string character, D-1t

QUIT command in editing, 3-16

Quitting

- EDT editor, 3-16

LSE, 3-17
VAXTPU editor, 3-17
Quotation marks
 around field values, 14-5
 edit string characters, D-1t
 enclosing literal values, 18-3
 using in prompting value expressions, 18-6

R

Rdb/ELN databases, 1-2
Rdb/VMS databases, 1-2
READ access mode, 7-21, 13-4t
READY command, 7-21, 13-1
 alias clause, 13-3
 for views, 15-15
Readying domains
 effect of, 13-3
 with the same name, 13-3
REAL data type, 9-13t
Recalling
 lines, 3-3
 lines versus commands and statements, 3-6
 previous command or statement, 3-5
Record definitions
 accessing using domain definitions, 10-1
 adding
 clauses to, 11-9
 fields to, 11-9
 group field names to, 11-9
 alphabetic fields in, 9-10
 changing
 field names in, 11-9
 field order in, 11-9
 size of fields in, 11-9
 column headers in, 9-7
 computed fields in, 9-15
 date fields in, 9-15
 deleting, 7-12
 editing, 9-24
 ending, 9-4

FILLER fields in, 9-8
formatting field values in, 9-20
improving readability of, 9-5
in domain definitions, 10-3
level numbers in, 9-3
naming conventions for, 9-5, 9-7
numeric fields in
 PIC clause, 9-11
PIC clause in, 9-9
query names in, 9-7
redefining fields in, 9-16
repeating fields in, 9-17
scaled values in, 9-11
top-level fields in, 9-4, 9-6
Record Management Services
 See RMS
Record selection expressions (RSEs),
 18-2
 and loops, 15-9
 FIND statement, 14-4
 FOR statement, 15-7 to 15-9
 MODIFY statement, 15-7 to 15-9,
 16-13
 naming groups of records, 14-4
 PRINT statement, 15-7 to 15-9
 REPORT statement, 20-4
 restricting
 fields from records, 14-4
 number of records, 14-4
 records from source, 14-4
 sorting records in, 14-4
 specifying records from more than
 one source, 14-4
Records
 accessing subsets of fields in, 15-14
 defining top-level fields in, 9-4
 displaying, 1-21
 modifying, 1-22
 storing, 1-20
RECOVER argument, EDIT command, 3-8
REDEFINE command, 9-24
REDEFINES clause, 9-16
 variable definition restriction, 18-4
REDUCE statement, 14-10

- advantages of using, 15-20
- disadvantages of using, 15-20
- REDUCED TO clause of RSE, 14-4
- Relational databases, 1-2, 1-30
- Relational operators, 18-11t
- RELEASE command, to remove collections, 14-16
- Remote data
 - See Distributed data
- RENAME command (DCL), 2-16
- Renaming files, 2-16
- Repeat count
 - for edit string characters, D-1t
 - for picture string characters, 9-9
- REPEAT statement
 - in store operations, 16-2
 - results, 17-7
- Repeating fields, 9-17
 - disadvantages of, 1-30
- REPORT statement
 - ON clause option, 20-4
 - RSEs in, 20-4
- Reports
 - control groups in
 - headers for, 20-13
 - summaries for, 20-13
 - controlling
 - column headers in, 20-6t
 - date displays in, 20-6t
 - detail lines in, 20-10 to 20-12
 - names of, 20-6t
 - page numbers in, 20-6t
 - page width of, 20-6t
 - size of, 20-6t
 - correcting mistakes in, 20-19
 - creating
 - detail lines for, 20-2
 - in procedures, 20-3
 - title pages for, 20-16
 - default settings for, 20-6t
 - ending, 20-2, 20-19
 - maximum settings for, 20-6t
 - prompting for values in, 20-6
 - required statements in, 20-5
 - sorting records for control groups, 20-13
 - specifying
 - names for, 20-2
 - page breaks in, 20-14
 - summarizing data in, 20-2
 - writing to printers and files, 20-4
- Response time
 - See Performance
- Restricting data access
 - by creating a view, 15-14
- Restructuring domains, 11-7
 - back up for, 11-8, 11-11
 - to change
 - file organization, 11-7
 - keys for indexed files, 11-7
 - record definition, 11-9
 - when changes do not require, 11-7, 11-9
- RETURN key
 - as line terminator, 1-3
 - when to press the, 5-4
- RMS, 1-30
 - See Record Management Services (RMS)
 - messages from, 2-9
- RSEs
 - See Record selection expressions
- RUNNING COUNT statistical operator, 18-17
- RUNNING TOTAL statistical operator, 18-17
- RW> prompt, 20-2

S

- S (sign) picture string character, 9-10t
- Sample data, copying with
 - NEWUSER program, 1-6
- SCALE clause, 9-11
 - with USAGE REAL fields, 9-14
- Scientific notation
 - specifying, D-1t
- Screen width, setting, 19-2

SEARCH command (DCL), 2-17
 Security methods
 See Protecting
 See also Editing
SEE privilege, 7-17
SELECT statement
 advantages of using, 15-20
 disadvantages of using, 15-20
 naming collection in, 14-9
 options
 FIRST, 14-8
 LAST, 14-8
 NEXT, 14-8
 NONE, 14-9
 PRIOR, 14-8
 record position number, 14-9
 WITH clause, 14-9
 restricting records in, 14-9
 uses for, 14-7
 Selected record
 releasing control over, 14-9
 restrictions for the, 14-9
 that cannot be found, 16-6
 Semicolon (;)
 at end of DELETE command, 7-12
 following hyphen (-), 7-9
 in path names, 7-5
 Separators, 4-2t, 4-5
 Sequential files
 advantages of, 11-5
 cannot erase records from, 11-6
 effect on performance, 11-6
SET ABORT command, 17-14
 in procedure to store records, 16-3
SET COLUMNS_PAGE command,
 19-2
SET DEFAULT command (DCL), 1-5
SET DICTIONARY command, 7-9
 access privilege requirements, 7-21
SET EDIT_BACKUP command,
 7-10t
SET NO PROMPT command, 5-5
SET SEMICOLON command, 5-6
SET statement (Report Writer)
 COLUMNS_PAGE, 20-6t
 DATE, 20-6t
 defaults, 20-6t
 LINES_PAGE, 20-6t
 MAX_LINES, 20-6t
 MAX_PAGES, 20-6t
 NO COLUMN_HEADER, 20-6t
 NO DATE, 20-6t
 NO NUMBER, 20-6t
 NO REPORT_HEADER, 20-6t
 NUMBER, 20-6t
 options, 20-6t
 REPORT_NAME, 20-2, 20-6t
SET VERIFY command, 17-16
 Setting screen width, 19-2
SHARED access option, 13-4t
SHOW ALL command, 7-10t
SHOW collection-name command,
 7-10t
SHOW COLLECTIONS command,
 7-10t
SHOW CURRENT command, 7-10t
SHOW DEFAULT command (DCL),
 2-20
SHOW DICTIONARIES command,
 7-10t
SHOW DICTIONARY command, 1-8,
 7-10t
SHOW DOMAINS command, 1-9,
 7-10t
SHOW EDIT command, 7-10t
SHOW FIELDS command, 1-12, 7-
 10t, 14-5
SHOW LOGICAL command (DCL),
 2-26
SHOW path-name command, 7-10t
 access privilege requirements, 7-21
SHOW PROCEDURES command, 7-
 10t, 17-2
SHOW PROTECTION command
 (DCL), 2-22
SHOW READY command, 7-10t
 displaying readied domains, 13-1
 displaying tables in workspace, 12-7

- for views, 15-15
- SHOW RECORDS command, 1-9, 7-10t
- SHOW SET_UP command, 7-10t
- SHOW SYMBOL command (DCL), 2-26
- SHOW SYNONYMS command, 7-10t
- SHOW TABLES command, 7-10t
- SHOW VARIABLES command, 7-10t
- SHOWP command
 - access privilege requirements, 7-21
- Signs
 - minus (-), 9-20
 - plus (+), 9-20
- SKIP print list element, 19-11
 - in reports, 20-14
- Slash (/)
 - arithmetic operator (division), 18-7
 - edit string character, D-1t
- Sort order, C-1t
- SORT statement, 14-12
 - advantages of using, 15-20
 - disadvantages of using, 15-20
- SORTED BY clause of RSE, 14-4
- Sorting records
 - in RSEs, 14-4
 - SORT statement, 14-12, 15-20
- SPACE print list element, 19-11
- Spaces
 - in edit strings, D-1t
 - needed around minus sign, 18-7
 - replacing hyphens with, 4-5
 - suppressing trailing, 17-2, 19-7
 - to separate comma and edit string, 4-5
- Specifying object types, EDIT command, 3-7
- Staged output, D-1t
- Starting
 - ADT, 1-18
 - DATATRIEVE
 - with logical symbol, 5-1
 - with options you choose, 5-2
 - DATATRIEVE Help, 6-1
 - DATATRIEVE Report Writer, 20-4

- Guide Mode, 6-3
- STARTING WITH relational operator, 18-11t
- Startup command file, DATATRIEVE, 5-2
- Statements
 - compound
 - and collections, 15-4
 - BEGIN-END, 17-8
 - cannot execute command files, 17-18
 - CHOICE, 17-11
 - contents of, 17-5
 - FOR, 17-7
 - IF, 17-10
 - keywords that define, 17-5
 - REPEAT, 17-7
 - restrictions, 17-13
 - THEN, 17-8
 - using indentation in, 17-7
 - WHILE, 17-9
 - continuing over lines, 5-4 to 5-6
 - continuing with hyphen, 5-5
 - ending, 4-5, 5-6, 7-9
 - privileges needed to use, 7-19
- Statistical operators, 18-17t
- Statistical value expressions, 18-16
 - editing results of, 18-18
 - missing values and, 18-17
 - that do not require OF clause, 18-20
 - that require OF clause, 18-17
- STD_DEV statistical operator, 18-17
- Stopping displays
 - with CTRL/C, 1-12
 - with NO SCROLL key, 1-11
- Storage space
 - conserving with tables, 12-1
 - required by various data types, 9-13
- Storing records, 1-20
 - checking input when, 16-4
 - entering no data for field when, 16-4
 - one at a time, 16-2
 - readying domains for, 16-1

using procedures when, 16-2
with REPEAT statement, 16-2
Subdictionaries, 7-23
Subdirectories
 VMS, 2-20
SUM statement, 18-20
 column headers, 18-21
 edit strings, 18-21
 entering, 18-20
 including record count in, 18-22
SUPERSEDE clause, 11-6
Symbols
 creating logical, 2-26
 deleting, 2-26
 DTR32, 1-6
 global, 2-28
 local, 2-28
Synonyms, creating keyword, 5-2
Syntax diagrams, 4-1
 braces in, 4-2t, 4-4
 brackets in, 4-2t, 4-4
 breaking rules in, 4-3
 ellipsis in, 4-2t
 lowercase words in, 4-2t, 4-4
 punctuation in, 4-2t
 repeating parts of, 4-4
 uppercase words in, 4-2t, 4-4
SYS\$DISK logical name, 2-25
SYS\$ERROR logical name, 2-25
SYS\$INPUT logical name, 2-25
SYS\$OUTPUT logical name, 2-25
 and ON clause or statement, 19-13
System failure
 recovering files edited during, 3-8

T

T (text) edit string character, 19-9, D-1t
TAB key
 using when storing records, 16-4
TAB print list element, 19-11
Tables, 1-26
 access privileges for invoking, 7-21
 associating values with, 12-1

creating, 1-24
 dictionary, 12-2 to 12-3
 domain, 12-3 to 12-4
displaying names of, 12-3
editing definitions of, 12-7
for validating field values, 12-8
privileges needed, 12-5
referencing with
 IN and NOT IN clauses, 12-5
 VALID IF clause, 12-4
 VIA clause, 12-5
specifying
 display format for values from, 12-2, 12-4
 headers for values from, 12-2, 12-4
types of DATATRIEVE, 1-25, 12-1
using in computed fields, 9-15
validating values with, 12-1

Terminals

See also VT100 terminal
See also VT200 terminal
See also VT52 terminal
getting ready to log in, 2-2

Text fields, 9-10

THEN statement, 17-8
 declaring variables in a, 18-5

Time

defining fields to store, 9-15
including in auditing information, 16-15
size of fields that store, 9-15

TJL file type, 3-8

TODAY value expression, 18-23

TOMORROW value expression, 18-23

Top-level fields, 9-6

TOTAL statistical operator, 18-17

Transaction file, using to modify records, 16-14

TYPE command (DCL), 2-14

U

UFD directories, 2-18

UIC

See User identification code
Update file, using to modify records,
16-14
UPDATE privilege, 7-17
Uppercase words in syntax diagrams,
4-2t, 4-4
USAGE clause
DATE option, 9-15
DISPLAY default for, 9-12
User identification code, 2-21
User name, 2-2
User-defined names and keywords, 4-4

V

V (decimal point) picture string character, 9-10t
VALID IF clause with tables, 12-8
Validating data
VALID IF clause of field definition,
9-23
VERIFY clause of MODIFY statement,
16-15
with tables, 12-1
with VERIFY clause of STORE statement,
16-4
Value expressions, 18-1, 18-2t
arithmetic, 18-7
concatenation, 19-8
conditional, 18-8
date, 18-23
FORMAT, 18-10
literals, 18-3
prompting, 18-6
record field name, 18-4
statistical, 18-16
variable field name, 18-4
Variable occurrence lists, 9-18
in sequential files, 11-6
Variables
computed, 9-16
computed by date values, 18-26
defining, 18-4 to 18-5
global, 18-5
local, 18-5

using date, 17-10
using in reports, 20-10, 20-14
VAX Text Processing Utility
See VAXTPU editor
VAXTPU editor, 3-1, 3-11
EDT Keypad Emulator, 3-11
exiting, 3-17
Extensible VAX Editor (EVE), 3-11
invoking from DATATRIEVE, 3-12
journal files, 3-8
quitting, 3-17
VERIFY clause
in a MODIFY statement, 16-15
of STORE statement, 16-4
Versions
of CDD objects, 1-9, 7-6
of files, 1-4, 2-12
of VAX DATATRIEVE, 1-8
Vertical bars
See Concatenation characters
VIDA databases, 1-2
Views
access privileges needed, 15-19
accessing fields in multiple domains,
15-17 to 15-19
cannot store records in, 15-14
containing subset of fields, 15-14 to
15-16
displaying
data in, 15-15, 19-11
definitions of, 15-15
erasing records in, 16-5
modifying records in, 16-11
reasons for creating, 15-14
Virtual fields, 9-15
VT100 terminal
resuming displays on, 2-7
suspending displays on, 2-7
VT200 terminal
resuming displays on, 2-7
suspending displays on, 2-7
VT52 terminal
resuming displays on, 2-7
suspending displays on, 2-7

W

W (day letter)

edit string character, D-1t

WHILE statement, 17-9

Boolean expressions in, 17-9

in a modify operation, 15-5

Wildcard character (*), 2-12

in RENAME command, 2-16

WITH clause

RSE, 14-4

SELECT statement, 14-9

WORD data type, 9-13t

WRITE access mode, 7-21, 13-4t

X

X (alphanumeric)

edit string character, D-1t

picture string character, 9-10t

Y

Y (year)

edit string character, D-1t

YESTERDAY value expression, 18-23

Z

Z (numeric) edit string character, D-1t

Zero (0)

edit string character, D-1t

Zeros, ensuring storage of leading,
16-3

ZONED data type, 9-13t



How to Order Additional Documentation

If you live in:	Call:	or Write:
New Hampshire, Alaska	603-884-6660	Digital Equipment Corp. P.O. Box CS2008 Nashua, NH 03061-2698
Continental USA, Puerto Rico, Hawaii	1-800-258-1710	Same as above.
Canada (Ottawa-Hull)	613-234-7726	Digital Equipment Corp. 940 Belfast Road Ottawa, Ontario K1G 4C2 Attn: P&SG Business Manager or approved distributor
Canada (British Columbia)	1-800-267-6146	Same as above.
Canada (All other)	112-800-267-6146	Same as above.
All other areas	—	Digital Equipment Corp. Peripherals & Supplies Centers P&SG Business Manager c/o DIGITAL's local subsidiary

Note: Place prepaid orders from Puerto Rico with the local DIGITAL subsidiary (phone 809-754-7575).

Place internal orders with the Software Distribution Center, Digital Drive, Westminister, MA 01473-0471.



Reader's Comments

VAX DATATRIEVE
Handbook
AA-W675B-TE

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)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Completeness (enough information)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Clarity (easy to understand)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Organization (structure of subject matter)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Figures (useful)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Examples (useful)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Index (ability to find topic)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Page layout (easy to find information)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

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 _____

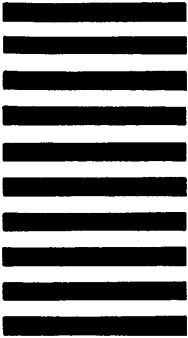
Phone _____

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

Cut Along Dotted Line



Reader's Comments

VAX DATATRIEVE
Handbook
AA-W675B-TE

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)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Completeness (enough information)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Clarity (easy to understand)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Organization (structure of subject matter)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Figures (useful)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Examples (useful)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Index (ability to find topic)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Page layout (easy to find information)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>



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™

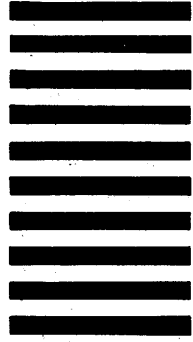


No Postage
Necessary
if Mailed
in the
United States

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

Cut Along Dotted Line