DATATRIEVE-11

digital

Call Interface Manual

Order Number: AA-U050C-TC

DATATRIEVE–11
Call Interface Manual

Order Number: AA–U050C–TC

July 1989

This manual explains how to use the DATATRIEVE–11 Call Interface to call DATATRIEVE from within programs written in high-level languages. It also explains how to use the DATATRIEVE–11 Remote Terminal Interface.

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

digital equipment corporation
maynard, massachusetts

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

The 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 Equipment Corporation or its affiliated companies.

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

The following are trademarks of Digital Equipment Corporation:

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

CID # 76330

ZK5066

Contents

Preface vii

Chapter 1 Remote DATATRIEVE–11: Call Interface and Terminal Interface

1.1 Interactive DATATRIEVE–11 1–2

1.2 The DATATRIEVE Distributed Server 1–3

1.3 The DATATRIEVE Local Server 1–3

1.4 The DATATRIEVE Remote Terminal Interface 1–3

1.5 The DATATRIEVE–11 Call Interface 1–5

Chapter 2 Using the DATATRIEVE–11 Remote Terminal Interface

2.1 Testing DATATRIEVE 2–2

2.2 Copying Domains 2–3

Chapter 3 Running Programs That Call DATATRIEVE

3.1 Compiling Your Program 3–2

3.2 Task Building 3–4

3.2.1 The DATATRIEVE Call Interface Object Module Library 3–8

3.2.2 Logical Unit Numbers and Event Flag Numbers 3–9

3.3 Overlays 3–12

Chapter 4 Writing Programs that Call DATATRIEVE–11

4.1 Overview of the Call Interface 4–1

4.2 DATATRIEVE States 4–7

4.3 Declaring the DATATRIEVE Access Block (DAB) 4–8

4.4 DATATRIEVE–11 Routines 4–10
4.4.1 Initializing the DATATRIEVE Interfaces 4–11
4.4.2 Passing Commands to DATATRIEVE (DTCMD) 4–13

4.5 Transferring Data 4–15
4.5.1 Retrieving Print Lines (DTLINE) 4–15
4.5.2 Retrieving Messages (DTMSG) 4–16
4.5.3 Passing Values to DATATRIEVE (DTPVAL) 4–20

4.6 Transferring Records 4–22
4.6.1 Defining Ports 4–23
4.6.2 Retrieving Records from DATATRIEVE (DTGETP) 4–24
4.6.3 Passing Records to DATATRIEVE (DTPUTP, DTPEOF) 4–26

4.7 Stopping the Execution of Commands 4–30

4.8 Closing the Call Interface 4–31

Chapter 5 Sample FORTRAN Programs

5.1 Creating an End-User Interface 5–1

5.2 The Main Program: MENU 5–3

5.3 The ESTABLISH Subroutine 5–5

5.4 The DISPLAY Subroutine 5–8

5.5 The SORT Subroutine 5–8

5.6 The MODIFY Subroutine 5–10

5.7 The REPORT Subroutine 5–13
5.8 The STORE Subroutine 5–16
5.9 The CHOOSE Subroutine 5–17
5.10 The PROMPT Subroutine 5–18
5.11 The CLSCRN Subroutine 5–18

Chapter 6 Sample COBOL Programs

6.1 Creating an End-User Interface 6–1
6.2 A Sample Payroll Application 6–5

Chapter 7 Sample BASIC Programs

7.1 Formatting a Report 7–1
7.2 Calculating a Linear Regression Equation 7–5

Chapter 8 Reference Section

8.1 DATATRIEVE Access Block 8–1
8.1.1 DATATRIEVE–11 States 8–3
8.1.2 Error Codes and Error Severity 8–4
8.1.3 Flags 8–5
8.1.4 The String Buffer 8–6

8.2 DATATRIEVE–11 Routines 8–7
DTCMD 8–9
DTCONT 8–12
DTFINI 8–14
DTGETP 8–15
DTINIT 8–17
DTLINE 8–21
DTMSG 8–22
DTPEOF 8–24
DTPUTP 8–26

DTPVAL 8–28
DTUNWD 8–30

Appendix A Definitions of the DATATRIEVE Access Block

A.1 FORTRAN–77 . . . . . A–1

A.2 COBOL–81 . . . . . A–2

A.3 BASIC-PLUS-2 . . . . . A–3

Index

Figures

1–1 The DATATRIEVE–11 Remote Terminal Interface 1–4
1–2 The DATATRIEVE–11 Call Interface 1–6
3–1 Allocating Logical Unit Numbers 3–11
3–2 Default Logical Unit Numbers 3–12
4–1 The DATATRIEVE Port 4–22
8–1 Argument List for DATATRIEVE–11 Routines 8–8

Tables

4–1 The DATATRIEVE Access Block 4–8
4–2 DATATRIEVE–11 Routines 4–11
8–1 The DATATRIEVE Access Block 8–1
8–2 The DATATRIEVE States 8–3
8–3 The DATATRIEVE Error Severity Codes 8–5
8–4 The Flags Field of the DATATRIEVE Access Block 8–5
8–5 Contents of the DAB$V_STRING Field 8–6
8–6 DTINIT Options 8–19

Preface

This manual explains how to call DATATRIEVE from within programs written in high-level languages such as FORTRAN, BASIC, and COBOL. It also explains how to use the Remote Terminal Interface to run DATATRIEVE on another node as an interactive process.

Intended Audience

This book addresses experienced users of at least one programming language. A knowledge of DATATRIEVE commands and statements is also required.

Structure

This book contains eight chapters and one appendix:

Chapter 1	Provides an introduction to the components of DATATRIEVE–11: Interactive DATATRIEVE–11, the DATATRIEVE–11 Distributed Server, the DATATRIEVE–11 Call Interface, and the DATATRIEVE–11 Remote Terminal Interface.
Chapter 2	Describes the Remote Terminal Interface and how to use it.
Chapter 3	Explains how to compile, task build, and run programs that use the DATATRIEVE–11 Call Interface.
Chapter 4	Describes the Call Interface and how to use it to call DATATRIEVE from within programs written in languages such as FORTRAN, COBOL, and BASIC.
Chapter 5	Contains sample FORTRAN programs.

Chapter 6	Contains sample COBOL programs.
Chapter 7	Contains sample BASIC programs.
Chapter 8	Is a reference section, describing each element of the DATATRIEVE Access Block and the DATATRIEVE Call Interface separately.
Appendix A	Lists example definitions of the DATATRIEVE Access Block in FORTRAN, COBOL, and BASIC.

Related Manuals

For more information about the subjects discussed in this book, consult the following manuals:

DATATRIEVE–11 User's Guide

DATATRIEVE–11 Reference Manual

DATATRIEVE–11 Installation Guide

The language reference manuals for FORTRAN–77, COBOL–81, and BASIC-PLUS-2 are also recommended for reference purposes.

Conventions

Programming examples and examples of the DATATRIEVE Remote Terminal Interface are printed in a dot matrix typeface. The DATATRIEVE or program output lines displayed on your terminal are printed in black. The commands and statements you enter from your terminal are printed in color.

Symbols and conventions used in syntax formats:

Convention	Meaning
UPPERCASE WORDS	Uppercase words are DATATRIEVE keywords.
lowercase words	Lowercase words indicate entries you must provide.
RET	This symbol indicates the RETURN key.
TAB	This symbol indicates the TAB key.
Color	Color in examples shows user input.
{ }	Braces mean you must choose one, but no more than one, of the enclosed entries.
[ ]	Brackets mean you have the option of choosing one, but no more than one, of the enclosed entries.
. . .	A horizontal ellipsis means you have the option of repeating the preceding element of the syntax format.
. . .	A vertical ellipsis in an example means that repetitious or irrelevant output has been omitted.
< >	Angle brackets mean the argument is an ASCII character string. These arguments can be passed by descriptor or as an address and length, depending upon the program language.

Chapter 1

Remote DATATRIEVE–11: Call Interface and Terminal Interface

This manual describes how to use:

- The DATATRIEVE–11 Remote Terminal Interface
- The DATATRIEVE–11 Call Interface

The **Remote Terminal Interface** enables you to run DATATRIEVE as an interactive process on another DECnet node. Thus, if you are logged on to a PDP–11 system, you can run DATATRIEVE on another node by typing RUN $REMDTR.

The **Call Interface** allows you to call DATATRIEVE from a program written in a high-level language. There is a Remote Call Interface and a Local Call Interface.

Using the Remote Call Interface, your program uses DECnet to call DATATRIEVE–11 running on your own PDP–11 system, or it can call DATATRIEVE running on another PDP–11 or VAX system linked to yours on the network.

The Local Call Interface calls DATATRIEVE–11 on your PDP–11 node by intertask communication, without using DECnet.

To understand the remote interface, you must understand the structure of DATATRIEVE as a whole.

DATATRIEVE–11 consists of the following components on your PDP–11 system:

- Interactive DATATRIEVE–11

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

- The DATATRIEVE–11 Distributed Server

  DDMF.TSK allows users on other DECnet nodes to use DATATRIEVE for accessing data files and data dictionaries on your node. That is, DDMF substitutes an interface to DECnet for the interface to the terminal in interactive DATATRIEVE.

- The DATATRIEVE–11 Local Server

  LCDDMF.TSK allows users to access data files and data dictionaries on the same PDP–11 node without interfacing with DECnet.

- The DATATRIEVE–11 Remote Terminal Interface

  REMDTR.TSK is an interactive program that uses the Remote Call Interface to communicate with the distributed server (on the local node or on a remote node). When you run REMDTR as a program, it looks as though you are running interactive DATATRIEVE on a remote node.

- The DATATRIEVE–11 Call Interface

  The DTCLIB.OLB object module library contains DATATRIEVE–11 subroutines that send commands to and receive information from the distributed or local server. Application programs can call these subroutines to access data files and data dictionaries on remote nodes.

The sections that follow describe these components in detail.

1.1 Interactive DATATRIEVE–11

When you type RUN $DTR on a PDP–11 system, you are running DTR.TSK, the interactive DATATRIEVE–11 task image. This program accepts DATATRIEVE–11 commands from the terminal and uses the terminal as the default output device. With DTR.TSK, you can use DATATRIEVE–11 commands and statements to access data stored in disk files as well as definitions stored in one of the data dictionaries on your system. The other books in this documentation set describe how to use interactive DATATRIEVE–11. You must understand how to use DATATRIEVE commands and statements before you can write programs that use the DATATRIEVE–11 Call Interface.

1.2 The DATATRIEVE Distributed Server

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

Both DATATRIEVE–11 and VAX DATATRIEVE have distributed servers.

The Remote Call Interface uses DECnet software to access the user's own PDP–11 or another node on the DECnet network. It then uses the DATATRIEVE Distributed Server (DDMF.TSK on PDP–11 systems or DDMF.EXE on VAX systems) to access data files and the Common Data Dictionary (CDD).

1.3 The DATATRIEVE Local Server

The Local DATATRIEVE Data Manipulation Facility (LCDDMF) is also called the DATATRIEVE Local Server. It is similar to the DATATRIEVE Distributed Server described previously, except that it is used only to communicate between programs and DATATRIEVE components on the same PDP–11 node. Using LCDDMF offers performance advantages; in addition, it makes DATATRIEVE data available on systems that do not have DECnet installed.

1.4 The DATATRIEVE Remote Terminal Interface

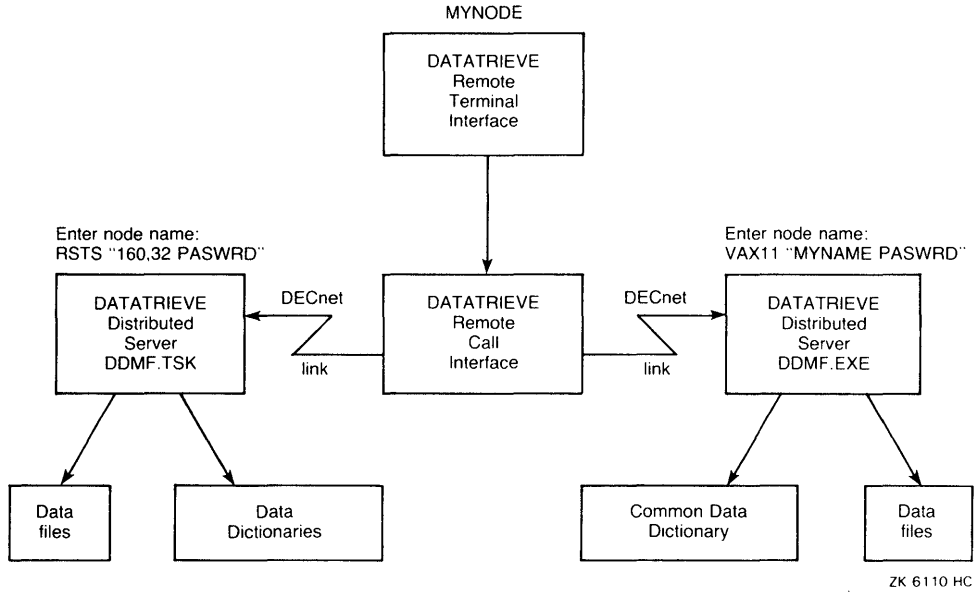
When you run REMDTR.TSK, it prompts you for a node name. You enter the node you wish to access and the user name or number and password of the account you want to use. The Terminal Interface uses the Remote Call Interface to establish a DECnet link to DDMF.TSK (on a PDP–11 system) or to DDMF.EXE (on a VAX system). When the link is established, REMDTR displays the banner identifying the version of DATATRIEVE being run and a special prompt: remDTR>. From this point on, you can type commands and statements, as though you were running interactive DATATRIEVE on the remote node.

NOTE

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

Figure 1–1 illustrates the use of the Remote Terminal Interface to access DATATRIEVE on a PDP–11 or VAX network node.

Figure 1–1: The DATATRIEVE–11 Remote Terminal Interface



Using the Terminal Interface gives you some advantages over using the SET HOST command to access another DECnet node:

- You can copy record and domain definitions from one node into a command file on another node, so that you can quickly set up identical domains on different nodes.
- You can copy data files or parts of data files from the remote node to the host node without leaving DATATRIEVE.
- You can use the Remote Terminal Interface to test statements and commands before including them in an application program that uses the Call Interface to access data across the network. For example, to see the default characteristics of DATATRIEVE on a particular node, you can run the Terminal Interface and type a SHOW command. Then you can use that information when writing the program.

Chapter 2 explains how to run REMDTR and perform these operations.

1.5 The DATATRIEVE–11 Call Interface

The DATATRIEVE–11 Call Interface consists of a set of routines contained in a library called DTCLIB.OLB. The Call Interface allows you to write high-level language programs that call DATATRIEVE, either on your own system or on another DECnet node.

To use the Call Interface, you include calls in your program to the external DATATRIEVE subroutines contained in the DTCLIB library. When you build the task image, you link the program to DTCLIB.OLB. The subroutines pass information between the calling programs and a local or remote DATATRIEVE Distributed Server. When you are running such a program, there are actually two task images active:

- Your program linked to DTCLIB.OLB
- DDMF (the DATATRIEVE Distributed Server) or LCDDMF (the DATATRIEVE Local Server) that has been activated to serve your program

There is a Local Call Interface and a Remote Call Interface. The Local Call Interface supports access to DATATRIEVE–11 on the same node (through LCDDMF and without using DECnet). The Remote Call Interface uses DECnet to access DATATRIEVE on any node in the network, including the node on which the task runs, through DDMF.

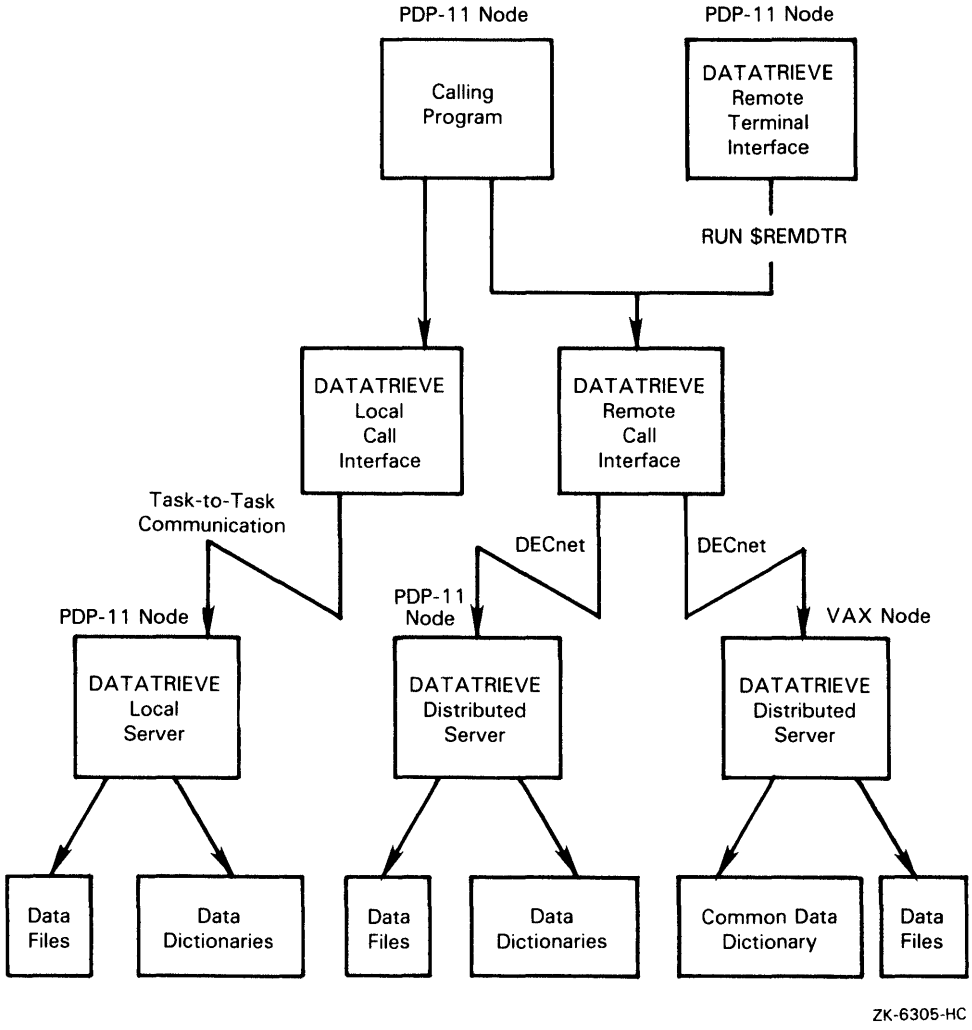
NOTE

You must have the DECnet software installed on your system before you can use the Remote Call Interface.

Micro/RSTS does not support the Remote Call Interface.

Figure 1–2 illustrates how your calling program interacts with components of DATATRIEVE.

Figure 1–2: The DATATRIEVE–11 Call Interface



Using the Call Interface extends the capabilities of DATATRIEVE in several ways:

- You can write programs to perform tasks that interactive DATATRIEVE cannot do for you. For example, your program can use DATATRIEVE to retrieve data and then have the program perform complex statistical

calculations, produce complicated reports, and customize the format of the terminal screen.

- Your program can customize the appearance of DATATRIEVE. For instance, you can build a menu that allows users to use DATATRIEVE without knowing its syntax. Some examples of menu-driven DATATRIEVE appear in the examples in Chapters 5, 6, and 7 of this book.
- Your program can access data through DATATRIEVE–11 on your own PDP–11 node, using either:
  - The Local Call Interface task-to-task communication without DECnet
  - The Remote Call Interface with DECnet
- Your program can access data through DATATRIEVE on other DECnet nodes (using the Remote Call Interface).

The Call Interface also extends the capabilities of programming languages. For example, you can input commands and record selection expressions for DATATRIEVE while your program is running. This means that records can be selected when the program runs, rather than when you write it. DATATRIEVE knows how the data file is organized and automatically searches for the records in the most efficient way.

The Call Interface also allows your program to use DATATRIEVE tables and procedures. For example, several programs can use a single table stored in the data dictionary. You can also use DATATRIEVE to validate data on input.

Chapters 3 through 8 in this manual tell you how to write application programs that use the Call Interface.

Chapter 2

Using the DATATRIEVE–11 Remote Terminal Interface

The command for running the Terminal Interface is the same on RSX–11M/M–PLUS or RSTS/E systems. Simply type RUN $REMDTR. If you get an error message, check with the system manager to make sure the program is installed.

You will be asked to choose a DECnet node by typing a node specification. At this point, you can simply type a node name or you can type a complete specification, with user name or account number and password:

```
Enter node name: MYVAX"MYNAME PASWRD"

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

When you use the complete form of the command, DECnet logs you in to the account named. If you simply use the node name, you are logged in to the default DECnet account. You may not be able to access the correct data files or dictionaries from this account.

After REMDTR logs in successfully, you can use DATATRIEVE on the target node just as you would interactively use DATATRIEVE on that node.

Here is an sample session:

```
>RUN $REMDTR RET
Enter node name: BADGER"USER PASWRD" RET

VAX Datatrieve V4.0
DEC Query and Report System
Type HELP for help
remDTR> READY YACHTS RET

Statement completed successfully.
remDTR> PRINT FIRST 1 YACHTS RET
```

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

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

Statement completed successfully.
remDTR>
```

The default directory is now [USER] on node BADGER. For example, if you type PRINT YACHTS ON BOATS.DAT, DATATRIEVE creates a file BOATS.DAT on BADGER, the remote system, in [USER], the default directory.

When you type EXIT or CTRL/Z to end the Terminal Interface session, remote DATATRIEVE prompts again for a node name. You can then choose another system or press CTRL/Z again to exit.

```
remDTR> CTRL/Z
Enter node name: CTRL/Z
```

2.1 Testing DATATRIEVE

When you are writing a program that uses the DATATRIEVE–11 Call Interface, you often need to find out the characteristics of a version of DATATRIEVE running on a remote node beforehand. The Terminal Interface can be useful in running this kind of test.

For example, assume that you are going to write a program that activates the DATATRIEVE–11 Distributed Server on a PDP–11 node named ELEVEN, using the account of a user named LITELLA. The program will do a store operation into the PERSONNEL domain. You want to test the interface to determine several things:

- Can my program initialize DATATRIEVE–11 using the user name and password on hand?
- What is the default data dictionary for LITELLA on ELEVEN?
- Is the data file in the correct directory on ELEVEN?
- Does LITELLA have sufficient privileges to store data in PERSONNEL?

You could use the Terminal Interface to answer these questions by following a sequence like the following:

```
>RUN $REMDTR [RET]
Enter node name: ELEVEN"LITELLA FZZBAL" [RET]

DATATRIEVE-11, DEC Query and Report System
Version: V03.03, 19-MAY-89
Type HELP for help
remDTR> SHOW DICTIONARY [RET]
The current dictionary is DB0:[100,120]QUERY.DIC;3

remDTR> SHOW DOMAINS [RET]
Domains:
        OWNERS          PERSONNEL       PHONES          SYNONYMS
        UPDATES         WORKSPACE       YACHTOWNERS     YACHTS
        YEAR_TO_DATE_COST

remDTR> SHOWP PERSONNEL [RET]

        3,UIC, [*,*], "R"

remDTR> READY PERSONNEL WRITE [RET]

remDTR> PRINT FIRST 4 PERSONNEL [RET]

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

00012 EXPERIENCED CHARLOTTE  SPIVA      TOP  12-Sep-72   $7,500  00012
00891 EXPERIENCED FRED       HOWL      F11   9-Apr-76  $59,594  00012
02943 EXPERIENCED CASS       TERRY     D98   2-Jan-80  $29,908  39485
12643 TRAINEE     JEFF       TASHKENT  C82   4-Apr-81  $32,918  87465

remDTR> EXIT [RET]
Enter node name: [CTRL/Z]
>
```

On a RSTS/E system you must specify the account number rather than name in your node specification.

2.2 Copying Domains

This section shows you how to copy a domain definition, record definition, and data file from a VAX system to a PDP–11 system using the Remote Terminal Interface.

The alternative is to use the DECnet network file copy utility. However, VAX DATATRIEVE uses the major-minor allocation rule by default, while DATATRIEVE–11 uses the left-right allocation rule. Therefore, if you use COPY, the fields of your records may not be aligned correctly on the PDP–11 node. Using the Remote Terminal Interface avoids this problem, because it allows DATATRIEVE to handle the allocation. See the *DATATRIEVE–11 Reference Manual* (the ALLOCATION clause) for more information on allocation.

The process involves the following steps:

1. From a PDP–11 system, use REMDTR to log into a VAX system. Use the EXTRACT command to pull the definitions from the Common Data Dictionary (CDD) and put them in a command file on the PDP–11 system. For example, using the Remote Terminal Interface from the node ELEVEN, your EXTRACT command would look something like the following:

   ```
   EXTRACT YACHTS, YACHT ON ELEVEN"100,120 FZZBAL"::DB1:YACHT.CMD
   ```

2. End the remote session by typing EXIT or CTRL/Z. The Terminal Interface prompts you for a new node name. Type the name of the host (PDP–11) node.
3. At this point you may need to edit the command file to remove features in the record definition specific to VAX DATATRIEVE, so that DATATRIEVE–11 will accept it. You may also want to change the name of the data file in the domain definition.
4. You are now connected to the DATATRIEVE–11 data dictionary, so you can execute the command file to load the definitions.
5. After the definitions are in place, define the data file.
6. Go back to the VAX DATATRIEVE server on the VAX node to store the records. Since VAX DATATRIEVE can ready domains across the network, use the restructuring mechanism to copy the records from the VAX system to the PDP–11. If you want to copy only a subset of records, use a DATATRIEVE record selection expression to restructure the domain.

Here is a sample session. Assume that you want to copy the YACHTS domain from a VAX node named VACKS to ELEVEN, a PDP–11 system running RSX–11M–PLUS. You are logged in to ELEVEN:

```
>RUN $REMDTR RET
Enter node name:  VACKS"LITELLA BZZWRD" RET

VAX Datatrieve V4.0
DEC Query and Report System
Type HELP for help
remDTR> EXTRACT YACHT, YACHTS ON ELEVEN"LITELLA FZZBAL"::DB1:YACHT.CMD RET

Statement completed successfully.
remDTR> ! RET
remDTR> ! Exit the remote session with CTRL/Z.  The Remote RET
remDTR> ! Terminal Interface prompts you for a new node name. RET
remDTR> ! Enter the PDP--11 node: RET
remDTR> ! RET
remDTR> CTRL/Z
Enter node name: ELEVEN"LITELLA FZZBAL" RET
```

```
DATATRIEVE-11, DEC Query and Report System
Version: V03.03, 19-MAY-89
Type HELP for help
remDTR> ! RET
remDTR> ! Now that you are on ELEVEN, you can execute the RET
remDTR> ! command file to load the definitions and define the RET
remDTR> ! data file. Note that you may need to edit the VAX RET
remDTR> ! DATATRIEVE record definition to remove syntax (such as RET
remDTR> ! MISSING VALUE) that is not part of DATATRIEVE-11. RET
remDTR> ! RET
remDTR> @YACHTS.CMD RET
DELETE YACHT;
DEFINE RECORD YACHT USING
01 BOAT.
  03 TYPE.
     06 MANUFACTURER PIC X(10)
        QUERY_NAME IS BUILDER.
     06 MODEL PIC X(10).
  03 SPECIFICATIONS
     QUERY_NAME SPECS.
     06 RIG PIC X(6)
        VALID IF RIG EQ "SLOOP","KETCH","MS","YAWL".
     06 LENGTH_OVER_ALL PIC XXX
        VALID IF LOA BETWEEN 15 AND 50
        QUERY_NAME IS LOA.
     06 DISPLACEMENT PIC 99999
        QUERY_HEADER IS "WEIGHT"
        EDIT_STRING IS ZZ,ZZ9
        QUERY_NAME IS DISP.
     06 BEAM PIC 99.
     06 PRICE PIC 99999
        VALID IF PRICE>DISP*1.3 OR PRICE EQ 0
        EDIT_STRING IS $$$,$$$.

DELETE YACHTS;
DEFINE DOMAIN YACHTS USING YACHT ON YACHT.DAT;

remDTR> DEFINE FILE FOR YACHTS KEY = TYPE (NO DUP), RET
DFN>        KEY = MODEL (DUP, NO CHANGE), RET
DFN>        ALLOCATION = 30, SUPERSEDE RET

remDTR> !
remDTR> ! Now go back to VACKS to store the records.
remDTR> !
remDTR> CTRL/Z
Enter node name: VACKS"LITELLA BZZWRD" RET

VAX Datatrieve V4.0
DEC Query and Report System
Type HELP for help
remDTR> ! RET
remDTR> ! First ready the domains using the distributed capability of RET
remDTR> ! VAX DATATRIEVE. RET
remDTR> ! RET
remDTR> READY YACHTS AS OLD_YACHTS RET

Statement completed successfully.
remDTR> READY YACHTS AT ELEVEN"LITELLA FZZBLL" AS NEW_YACHTS WRITE RET
```

```
Statement completed successfully.
remDTR> ! RET
remDTR> ! Now use the restructuring mechanism to move the records. RET

remDTR> ! This version uses a record selection expression to move only RET
remDTR> ! a subset of the records. RET
remDTR> ! RET
remDTR> NEW_YACHTS = OLD_YACHTS WITH PRICE NOT MISSING RET

Statement completed successfully.
remDTR> ! RET
remDTR> FINISH RET

Statement completed successfully.
remDTR> CTRL/Z
Enter node name: ELEVEN"LITELLA FZZBLL" RET

DATATRIEVE-11, DEC Query and Report System
Version: V03.03, 19-MAY-89
Type HELP for help
remDTR> READY YACHTS RET

remDTR> PRINT COUNT OF YACHTS RET
  50

remDTR> CTRL/Z
Enter node name: CTRL/Z

>
```

Chapter 3

Running Programs That Call DATATRIEVE

This chapter and those that follow tell you how to write programs that call DATATRIEVE through the DATATRIEVE–11 Remote Call Interface. The examples in these chapters are written in the following languages:

- BASIC-PLUS-2—Version 2.4
- FORTRAN–77—Version 5.2
- COBOL–81—Version 2.4

You can use the Call Interface with previous versions of these languages, or with other languages developed by Digital. However, the examples in this book may use features that these other versions do not support. If you wish to copy the examples and use them with other languages, conversion may be necessary.

Running a program that calls DATATRIEVE requires the same steps as running any program:

- Create the source file
- Compile the program
- Build the executable task image
- Run the program

This chapter describes how to compile and task build programs using the DATATRIEVE–11 Remote Call Interface.

3.1 Compiling Your Program

Compile your program as you would any high-level language source file. The exact syntax for compiling depends on several factors:

- Your operating system (RSTS/E, RSX–11M, RSX–11M–PLUS)
- Your command language (MCR, DCL, CCR)
- Your high-level language (BASIC-PLUS-2, COBOL–81, FORTRAN–77, or some other PDP–11 language)

For example, if you are compiling a COBOL–81 program on an RSX–11M–PLUS system, using the MCR Command Language Interpreter, the command line for compilation would look like the following:

```
>C81 RET
C81>CSTORE.OBJ,CSTORE.LST=CSTORE.C81 RET
C81> CTRL/Z
>
```

In the previous example:

`CSTORE.OBJ`	Is the object file that the compiler creates.
`CSTORE.LST`	Is the source listing that the compiler creates.
`CSTORE.C81`	Is the input source file.

If you are using the DCL command language on a RSTS/E system, a typical compilation line would look like the following:

```
$ COBOL/C81 CSTORE.C81 RET
$
```

The sequence for compiling a FORTRAN–77 program is similar. The default command for invoking the FORTRAN–77 compiler is F77, although the system manager has the option of choosing a different 3-letter command. In the following example, the use of the compiler switch, /-SP, prevents the source listing from being spooled to the printer.

For example, if the FORTRAN–77 compiler has been installed on an RSX–11M or RSX–11M–PLUS system, using MCR, you might compile the program FSTORE.FTN as follows:

```
>F77 RET
F77>FSTORE,FSTORE/-SP=FSTORE RET
F77> CTRL/Z
>
```

On RSX–11M and RSX–11M–PLUS, you can also use the F77 command at MCR command level:

```
>F77 FSTORE,FSTORE/-SP=FSTORE RET
>
```

If the system manager has not installed the FORTRAN–77 compiler, it will not be resident in memory. This means that you must run the compiler like any other task. To do this, precede the compiler name with the RUN command and a dollar sign. The dollar sign tells RSX to look for the compiler in the system account. The compilation sequence might look like the following:

```
>F77 RET
MCR - Task not in system
>RUN $F77 RET
F77>FSTORE,FSTORE/-SP=FSTORE RET
F77> CTRL/Z
>
```

To use the BASIC-PLUS-2 compiler, you must enter the BASIC environment. In most cases, the command for doing this is:

```
> BP2 RET

PDP-11 BASIC-PLUS-2 V2.4-0

BASIC2
```

However, any 3-character name can be chosen for the compiler during installation. See your system manager for the name of the BASIC-PLUS-2 compiler on your system.

Once inside the BASIC environment, you must bring a copy of the source program into memory and issue the COMPILE command:

```
BASIC2
OLD SOURCE.B2S RET
BASIC2
COMPILE RET
BASIC2
```

For more information on how to invoke the compiler and compile your program, and for complete lists of compiler options, see the user's guide for your language and operating system.

3.2 Task Building

The Task Builder (TKB) is a system program that links object modules to form an executable task image. You invoke the Task Builder by entering the TKB command. Because you must link your object module or modules with several libraries and specify options to the Task Builder, it is easiest to place the list of input files and options in a Task Builder command file. Thus, the command to run the Task Builder for program PROG is:

```
TKB @PROG.CMD
```

A job can contain two types of calls: remote and local. The Remote Call Interface uses the DATATRIEVE Distributed Server across DECnet and can access DATATRIEVE databases on either VAX or PDP–11 nodes (including the host PDP–11 node). The Local Call Interface uses only task-to-task communication within the host PDP–11 node.

Note that a job can have any number of remote calls but only one local call. In addition, there are other restrictions specific to certain operating system environments that will be described later in this chapter.

The Task Builder command file must indicate whether the task uses the Remote Call Interface, the Local Call Interface, or both. For this reason, you may have to modify Task Builder command files in current use when you use this version of DATATRIEVE–11.

When you are building the task image for a program that uses the DATATRIEVE–11 Call Interface, your Task Builder command file looks like one of the following. The numbered comments explain its elements. Examples 1, 2, and 3 show tasks containing remote calls only for FORTRAN, BASIC-PLUS-2, and COBOL–81. Example 4 shows a task containing a local call only (using FORTRAN; BASIC and COBOL tasks are similar). Example 5 shows a task containing both remote and local calls.

1. GROUPE.CMD, a command file for the FORTRAN program GROUPE and a subroutine MESAGE that were compiled on a RSTS/E system.

```
GROUPE,GROUPE/-SP=GROUPE,MESAGE, (1)
LB:F4POTS/LB,                    (2)
LB:RMSLIB/LB,                    (3)

LB:DTCLIB/LB:CIFOR:NCRSTS:NOLC,  (4)

LB:DTCLIB/LB                     (5)
/                                (6)
UNITS = 6                        (7)
//                               (8)
```

❶ The user's input and output files. These include the executable task image file (.TSK) that the Task Builder creates, an optional map file, and two input object modules, the main program and the subroutine.

❷ The FORTRAN–77 object-time system (OTS) library. Sometimes the FORTRAN OTS is contained in the system object module library, SYSLIB.OLB. If so, this line is not necessary; the Task Builder will search SYSLIB and find the FORTRAN modules automatically. Your system manager can tell you whether F4POTS.OLB is installed separately or included in SYSLIB.OLB.

❸ The Record Management Services (RMS) object module library. This library is necessary if your FORTRAN program accesses a file or uses the WRITE or READ statements for terminal I/O.

❹ The FORTRAN–77 and RSTS/E modules from DTCLIB, the DATATRIEVE–11 Call Interface object module library. This line specifies three modules (not two as in previous versions). CIFOR is the FORTRAN–77 module; NCRSTS is the RSTS/E module for the Remote Call Interface; the NOLC module is needed because the task does not use the Local Call Interface. If the Local Call Interface were being used, the entry would specify the LCRSTS module instead of NOLC. For more information, see Section 3.2.1 on the DATATRIEVE–11 Call Interface object module library.

❺ The rest of DTCLIB.OLB. For more information, see Section 3.2.1.

❻ The single slash marks the beginning of a set of Task Builder options. Your run-time system may require that you specify some Task Builder options, but on RSTS systems the DATATRIEVE–11 Call Interface does not.

❼ Here, the command file specifies the number of logical unit numbers the program uses. Again, your program may require this entry in the command file; DATATRIEVE does not.

❽ The double slashes mark the end of the set of Task Builder options.

2. PROG.CMD, the Task Builder command file generated by the BUILD command in the BASIC-PLUS-2 environment on an RSX–11M–PLUS system. The command file has been edited to add references to the libraries. In addition, the /MP qualifier has been removed on the input object module to eliminate the search of the Overlay Description Language (ODL) file.

```
SY:PROG/CP=SY:PROG,MESAGE,PROMPT,CLSCRN,  ❶
LB:[1,1]BP2OTS/LB,                        ❷

LB:[1,1]DTCLIB/LB:CIBAS:NC11M:NOLC,       ❸

LB:[1,1]DTCLIB/LB                         ❹
/
UNITS = 15                                ❺
ASG = TI:13:15
ASG = SY:5:6:7:8:9:10:11:12
GBLPAT=PROG:LUNMAP:001700:000000          ❻
EXTTSK= 512
//
```

❶ The user's input and output files. MESAGE.OBJ, PROMPT.OBJ, and CLSCRN.OBJ are external subroutines that PROG calls.

❷ The BASIC-PLUS-2 object-time system library.

❸ The BASIC and RSX–11M/M–PLUS modules from DTCLIB.OLB, the DATATRIEVE–11 Call Interface object module library. This line specifies NC11M, the Remote Call Interface module for BASIC running on RSX–11M–PLUS. If the Local Call Interface were also being used, the line would specify the LC11M module; instead it specifies NOLC to indicate no local calls. For more information, see Section 3.2.1.

❹ The rest of the modules from DTCLIB.OLB.

❺ A Task Builder option, the number of logical units the program will use. BASIC automatically allocates these logical unit numbers (LUNs) and assigns them to the terminal and to the system, as the next two lines indicate.

❻ Another Task Builder option. This option specifies which LUNs the Call Interface can use. It is important that BASIC-PLUS-2 and DATATRIEVE–11 do not try to access the same LUNs. In some cases, therefore, you must determine which LUNs to allocate to the Call Interface and fill in the LUNMAP value. See Section 3.2.2 for more information on logical unit numbers.

3. PAYROL.CMD, the Task Builder command file for a COBOL program to be run on an RSX–11M–PLUS system.

```
PAYROL,PAYROL=PAYROL,ERSPGE               ❶
LB:[1,1]C81LIB/LB,                        ❷
LB:[1,1]RMSLIB/LB,                        ❸

LB:[1,1]DTCLIB/LB:CICOB:NC11M:NOLC,       ❹
```

```
LB:[1,1]DTCLIB/LB                          ❺
/
UNITS=10                                   ❻
GBLPAT=PAYROL:LUNMAP:177700:177777
//
```

❶ The user's input and output files. ERSPGE is a subroutine.

❷ The COBOL–81 object module library.

❸ RMSLIB is the object module library for RMS–11, the Record Management System. PAYROL.CBL opens files for reading and writing, so you must specify the RMS library in the command file.

❹ The COBOL and RSX–11M/M–PLUS modules from DTCLIB.OLB, the DATATRIEVE–11 Call Interface object module library. This line specifies the Remote Call Interface module but not the Local Call Interface, similar to the two previous examples. For more information, see Section 3.2.1.

❺ The rest of the modules from DTCLIB.OLB.

❻ The Task Builder options specify the number of logical units the program can use and which of those the Call Interface can use. See Section 3.2.2 for more information on logical unit numbers.

4. BUNCH.CMD, a command file for the FORTRAN program BUNCH and a subroutine MESAGE that were compiled on a RSTS/E system. This task uses the Local Call Interface but not the Remote Call Interface.

```
BUNCH,BUNCH/-SP,BUNCH=BUNCH,MESAGE,        ❶
LB:DTCLIB/LB:CIFOR:LCRSTS:NONC,            ❷
LB:F77RMS/LB,                              ❸
LB:RMSLIB/LB,                              ❹
LB:DTCLIB/LB,                              ❺
/
UNITS=10
GBLDEF=TF.CCO:0,TF.RNE:0
GBLPAT=BUNCH:LUNMAP:177700:177777
//
```

❶ The user's input and output files.

❷ The FORTRAN–77 and RSTS/E modules are called from DTCLIB. This task uses only the Local Call Interface. In addition to the CIFOR module, it specifies LCRSTS to indicate the use of local calls, and NONC to indicate that no remote calls are used.

❸ The FORTRAN–77 object-time system library.

❹ The RMS object module library. This library is necessary if your FORTRAN program accesses a file or uses the WRITE or READ statements for terminal I/O.

❺ The rest of DTCLIB.OLB. For more information, see Section 3.2.1.

5. BUNCH2.CMD, a command file for the FORTRAN program BUNCH2 and a subroutine MESAGE that were compiled on a RSTS/E system. This example is the same as Example 4 except for the second line; the difference occurs because this task includes both local and remote calls.

```
BUNCH2,BUNCH2/-SP,BUNCH2=BUNCH2,MESAGE,
LB:DTCLIB/LB:CIFOR:LC11M:NC11M,
LB:F77RMS/LB,
LB:RMSLIB/LB,
LB:DTCLIB/LB,
/
UNITS=10
GBLDEF=TF.CCO:0,TF.RNE:0
GBLPAT=BUNCH2:LUNMAP:177700:177777
//
```

3.2.1 The DATATRIEVE Call Interface Object Module Library

Every Task Builder command file must specify the object module library, DTCLIB.OLB. DTCLIB.OLB contains the modules necessary for the DATATRIEVE–11 Call Interface.

You must include in the command file a reference to DTCLIB.OLB as a whole. When you do, the Task Builder automatically searches for and uses the modules that your program calls for.

In each case, you must name three DTCLIB modules explicitly:

1. The call interface module for the programming language you are using. You specify one of the following:
   - CIBAS if using BASIC
   - CICOB if using COBOL
   - CIFOR if using FORTRAN
2. The appropriate module for the Local Call Interface. The selection of module depends on what operating system you are using and on whether the task uses the Local Call Interface. Specify one of the following:
   - LCRSTS if using the Local Call Interface on a RSTS/E or Micro/RSTS system
   - LC11M if using the Local Call Interface on an RSX–11M, RSX–11M–PLUS, or Micro/RSX system
   - NOLC if not using the Local Call Interface in the task

3. The appropriate module for the Remote Call Interface. The selection of module depends on what operating system you are using and on whether the task uses the Remote Call Interface. Specify one of the following:
   - NCRSTS if using the Remote Call Interface on a RSTS system
   - NC11M if using the Remote Call Interface on an RSX–11M, RSX–11M–PLUS, or Micro/RSX system
   - NONC if not using the Remote Call Interface in the task

Micro/RSTS is a special case; it supports only the Local Call Interface, not the Remote Call Interface. Therefore, only the following module specifications are valid on Micro/RSTS if the Local Call Interface is used:

- `LB:DTCLIB/LB:CIBAS:LCRSTS:NONC,`
- `LB:DTCLIB/LB:CICOB:LCRSTS:NONC,`
- `LB:DTCLIB/LB:CIFOR:LCRSTS:NONC,`

The order in which you specify the modules is not critical. For example, the following three lines are equivalent:

- `LB:DTCLIB/LB:CIBAS:LC11M:NC11M,`
- `LB:DTCLIB/LB:CIBAS:NC11M:LC11M,`
- `LB:DTCLIB/LB:NC11M:CIBAS:LC11M,`

Each specifies the BASIC call interface module, along with the Local and Remote Interface modules for the RSX–11M operating system.

If you are not using either the Local or Remote Call Interface, you do not need modules from DTCLIB. A module selection command such as the following is legal but does nothing useful:

```
LB:DTCLIB/LB:CIFOR:NOLC:NONC,
```

3.2.2 Logical Unit Numbers and Event Flag Numbers

When you are using RSX–11M/M–PLUS, your Task Builder command file must specify the number of logical unit numbers (LUNs) your task image will use. This may also be true on a RSTS/E system, depending on the run-time system.

In addition, on RSX–11M/M–PLUS systems, the Call Interface uses LUNs to perform DECnet services. For this reason, if you are running an RSX operating system, you must specify in the Task Builder command file which LUNs the Call Interface can use. Otherwise, the Call Interface and the language processor may try to use the same LUNs. This section describes

how to determine which LUNs are available, and how to assign them to the DATATRIEVE–11 Call Interface.

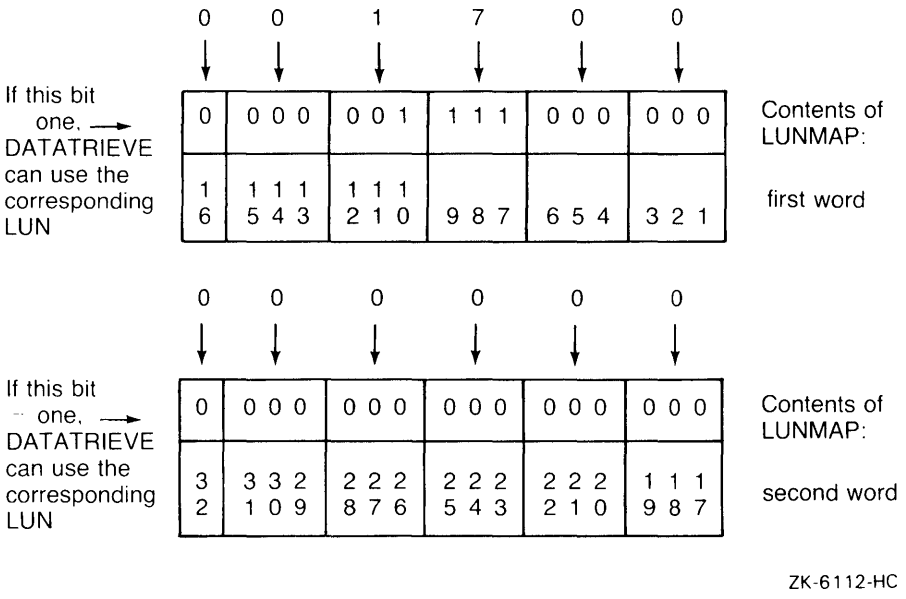
In the Call Interface, there is a 2-word global storage area called LUNMAP, which is used to specify exactly which LUNs DATATRIEVE can use. The same area is used to determine which event flag numbers are reserved for DATATRIEVE. When you have decided which LUNs are available, you use the Task Builder qualifier GBLPAT to map their numbers to LUNMAP. DATATRIEVE will also use the event flags associated with those numbers.

Following is an example of a command file that specifies LUNs:

```
PROG,PROG/-SP=PROG,MESAGE,
LB:[1,1]F4POTS/LB,
DTCLIB/LB:CIFOR:NC11M:LC11M,
DTCLIB/LB
/
UNITS=10
GBLPAT=PROG:LUNMAP:001700:000000
//
```

The UNITS = 10 qualifier specifies that a total of 10 logical unit numbers are allocated for this program. Assume that LUNs numbered 1 to 6 are reserved for the FORTRAN program. The Call Interface, therefore, can use LUNs 7 to 10. The GBLPAT option specifies the numbers of the LUNs that the Call Interface uses by mapping that value to the global symbol LUNMAP, as shown in Figure 3–1.

Figure 3–1: Allocating Logical Unit Numbers

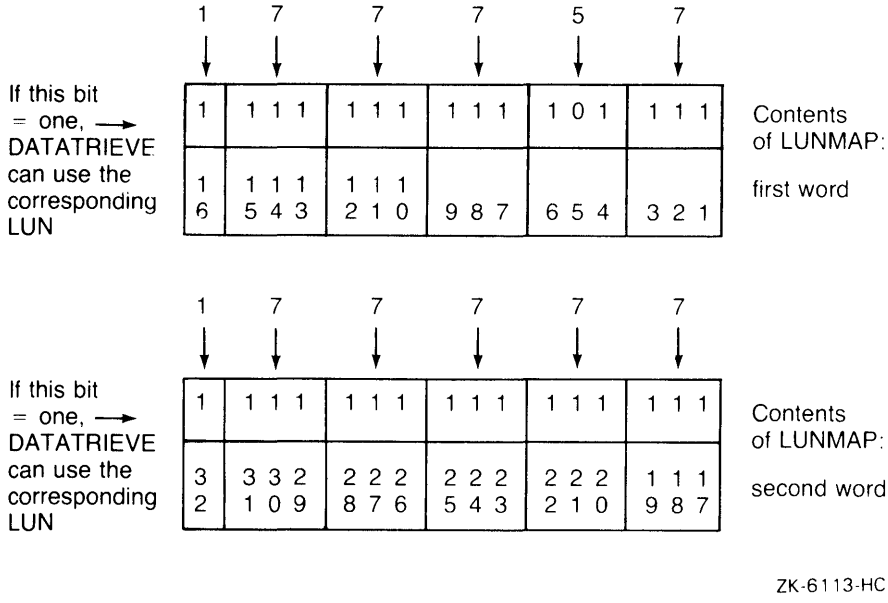


That is, only bits 7, 8, 9, and 10 are set in LUNMAP. Thus LUNs numbered 7, 8, 9, and 10 are reserved for use by the DATATRIEVE–11 Call Interface, and LUNs 1 to 6 can be used by FORTRAN. Similarly, the Call Interface will use event flag numbers 7 to 10.

To determine how many LUNs the Call Interface requires beyond those needed for the language processor, perform the following calculation. First, determine how many logical links your program will activate at once. That is, how many calls to the initialization routine DTINIT does your program make? Take this number and add 1. Thus, if your program contains only one call to DTINIT, you need two LUNs for the Call Interface, in addition to those required by your language.

If you do not specify the usage of LUNs, LUNMAP is set to 177757:177777 as Figure 3–2 shows.

Figure 3–2: Default Logical Unit Numbers



In this setting, only bit number 5 is clear, so all the LUNs and event flags except number 5 are available, up to the default limit set by the UNITS = qualifier for DATATRIEVE.

3.3 Overlays

You should overlay subroutines in a program that calls DATATRIEVE only if the subroutines do not contain calls to DATATRIEVE routines. For example, if you have a program that calls DATATRIEVE to obtain data and then calls a subroutine to perform calculations on that data, the subroutine could be overlaid without interfering with the DATATRIEVE Call Interface. For more information on the concepts underlying overlays, see your language user's guide or the Task Builder manual for your operating system.

Chapter 4

Writing Programs that Call DATATRIEVE–11

A program interacts with DATATRIEVE in much the same way as a user does. The program passes command strings, values, and structured records to DATATRIEVE, and DATATRIEVE passes messages, print lines, and structured records to the program.

This chapter describes the components of the DATATRIEVE–11 Call Interface and explains how to use the Call Interface in programs written in high-level languages such as BASIC, FORTRAN, and COBOL.

4.1 Overview of the Call Interface

Three components make up the DATATRIEVE–11 Call Interface:

- The DATATRIEVE Access Block (DAB)

  DATATRIEVE and your program use the DAB to communicate with each other. You set up storage for the DAB in your program, and DATATRIEVE uses it to return several pieces of information to your program, including:

  - The current DATATRIEVE state
  - A set of flags that DATATRIEVE uses to pass information to your program
  - Strings such as DATATRIEVE prompts and port names, and their length
  - A status code, which is either the success condition code or an error number if the routine did not complete successfully

- DATATRIEVE states

  After DATATRIEVE executes a command or statement, it enters a particular state and returns control to your program. A state is indicated by a value that DATATRIEVE stores in the DAB. Your program can test this value to see what routine DATATRIEVE expects you to call next.

- DATATRIEVE routines

  Your program passes control to DATATRIEVE by calling external routines. These routines allow you to execute DATATRIEVE commands and statements, pass and retrieve information, and handle error conditions.

Calling DATATRIEVE from a program involves the following steps:

1. Declare a DATATRIEVE Access Block (DAB).
2. Initialize the DATATRIEVE interface.
3. Check the DATATRIEVE state to see which routine DATATRIEVE expects you to call next.
4. Call DATATRIEVE routines to:
   a. Pass commands and statements
   b. Pass values and records
   c. Retrieve records
   d. Retrieve print lines and messages
5. Handle errors and display messages.
6. Close the interface.

This chapter includes simple examples to illustrate all the functions that the Call Interface performs. Chapters 5 through 7 contain more complete examples. To get you started, several simple programs follow, showing how the components of the Call Interface fit together in a program. BASIC, FORTRAN, and COBOL versions are included. Each program calls DATATRIEVE routines to: (1) initialize the interface on a local or remote node, (2) choose a dictionary, (3) ready a domain, and (4) print the domain.

FORTRAN–77

This FORTRAN example uses a subroutine (MESAGE) to print message and print lines. The subroutine appears in Section 4.5.2.

```
C
C Include definition of the DAB and declare variables.
C
        INCLUDE 'DAB11.FTN'
        CHARACTER*20 DOMAIN, DICT
        CHARACTER*31 NODE
        INTEGER*4    SEV
        INTEGER*4    LENGTH
C
C Prompt for a node name and initialize the interface.
C
        WRITE (5,100)
100     FORMAT (' Enter node: ',$)
        READ (5,1000) LENGTH, NODE
1000    FORMAT (Q,A)
        CALL DTINIT (DAB, STRLEN, BUFLEN, NODE, LENGTH, NOSEMI)
        CALL MESAGE (SEV)
C
C Choose a dictionary.
C
        WRITE (5,200)
200     FORMAT (' What dictionary would you like to use? ',$)
        READ (5,1000) LENGTH, DICT
        CALL DTCMD (DAB, 'SET DICTIONARY !CMD;', 20, DICT, LENGTH)
        CALL MESAGE (SEV)
C
C Ready the domain.
C
        WRITE (5,300)
300     FORMAT (' What domain would you like to use? ',$)
        READ (5,1000) LENGTH, DOMAIN
        CALL DTCMD (DAB, 'READY !CMD;', 11, DOMAIN, LENGTH)
        CALL MESAGE  (SEV)
C
C Print the domain.
C
        CALL DTCMD (DAB, 'PRINT !CMD;', 11, DOMAIN, LENGTH)
        CALL MESAGE  (SEV)
C
C Close the interface.
C
        CALL DTFINI (DAB)
        END
```

COBOL–81

In this COBOL–81 example, the 900-PRINT-MESSAGES paragraph performs the same function as the MESAGE subroutine in the FORTRAN example.

```
IDENTIFICATION DIVISION.
PROGRAM-ID.           PRINT.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.

**************************************
* Copy in the DATATRIEVE Access Block. *
**************************************

 COPY "DAB11.CBL".

01 MSGBUF    PIC X(80).
01 MSGLEN    PIC 9(4) COMP.
01 NODE      PIC X(30).
01 COMMAND   PIC X(80).
01 DICT      PIC X(30).
01 DOMAIN    PIC X(30).

PROCEDURE DIVISION.
010-INITIALIZE-INTERFACE.
        DISPLAY "Enter node: " WITH NO ADVANCING.
        ACCEPT NODE.
        CALL "DTINIT" USING DAB STRLEN BUFLEN
                BY DESCRIPTOR NODE
                BY REFERENCE NOSEMI.
        PERFORM 900-PRINT-MESSAGES UNTIL
                DAB-W-STATE NOT = DTR-K-STATE-MSG AND
                DAB-W-STATE NOT = DTR-K-STATE-LINE.

020-CHOOSE-DICTIONARY.
        DISPLAY "What dictionary would you like to use? " WITH NO ADVANCING.
        ACCEPT DICT.
        MOVE "SET DICTIONARY !CMD;" TO COMMAND.
        CALL "DTCMD" USING DAB
                        BY DESCRIPTOR COMMAND
                        DICT.
        PERFORM 900-PRINT-MESSAGES UNTIL
                DAB-W-STATE NOT = DTR-K-STATE-MSG AND
                DAB-W-STATE NOT = DTR-K-STATE-LINE.

030-READY-DOMAIN.
        DISPLAY "What domain would you like to use? " WITH NO ADVANCING.
        ACCEPT DOMAIN.
        MOVE "READY !CMD;" TO COMMAND.
        CALL "DTCMD" USING DAB
                        BY DESCRIPTOR COMMAND
                        DOMAIN.
        PERFORM 900-PRINT-MESSAGES UNTIL
                DAB-W-STATE NOT = DTR-K-STATE-MSG AND
                DAB-W-STATE NOT = DTR-K-STATE-LINE.
```

```
040-PRINT-DOMAIN.
        MOVE "PRINT !CMD;" TO COMMAND.
        CALL "DTCMD" USING DAB
                        BY DESCRIPTOR COMMAND
                        DOMAIN.
        PERFORM 900-PRINT-MESSAGES UNTIL
                DAB-W-STATE NOT = DTR-K-STATE-MSG AND
                DAB-W-STATE NOT = DTR-K-STATE-LINE.
        PERFORM 999-EOJ.

900-PRINT-MESSAGES.
        IF DAB-W-STATE = DTR-K-STATE-MSG
                CALL "DTMSG" USING DAB
                                BY DESCRIPTOR MSGBUF
                                BY REFERENCE MSGLEN.
                DISPLAY MSGBUF.
                IF DAB-W-ERR-SEV = SEV-K-SEVERE GO TO 999-EOJ.
        IF DAB-W-STATE = DTR-K-STATE-LINE
                CALL "DTLINE" USING DAB
                                BY DESCRIPTOR MSGBUF
                                BY REFERENCE MSGLEN.
                DISPLAY MSGBUF.
        CALL "DTCONT" USING DAB.

999-EOJ.
        CALL "DTFINI" USING DAB.
        STOP RUN.
```

BASIC-PLUS-2

The BASIC example uses the subroutine TEST_STATUS to check for errors and display messages and print lines.

```
100     %INCLUDE "DAB11.B2S"  ! DTR definitions file

        ! Declarations:
        !
        DECLARE WORD LENGTH
        COMMON (Buf) STRING MSGBUF = 80%,
                            COMAND = 80%,
                            NODE = 30%,
                            DICT = 30%,
                            DOMAIN = 30%

        ! Prompt for the node name and initialize the DATATRIEVE
        ! Distributed Server on that node.
        !
   Initialize_Interface:
        LINPUT "What node would you like to use"; NODE
        CALL DTINIT (DAB, STRLEN, BUFLEN, NODE, NOSEMI)
        GOSUB Test_status
```

```
     ! Prompt for a dictionary or CDD directory and use that value
     ! as a parameter to pass a command to DATATRIEVE.
     !
Choose_Dictionary:
     LINPUT "What dictionary would you like to use"; DICT
     COMAND = "SET DICTIONARY !CMD;"
     CALL DTCMD (DAB, COMAND, DICT)
     GOSUB Test_status

     ! Pass a SHOW command to show the domains.  Ask the user to
     ! choose one and ready it, using DTCMD to pass the command to
     ! DATATRIEVE.
     !
Ready_Domain:
     LINPUT "What domain do you want to use"; DOMAIN
     CALL DTCMD (DAB, "READY !CMD;", DOMAIN)
     GOSUB Test_status

     ! Pass a PRINT statement, using the domain name as a
     ! parameter.  The subroutine Test_status handles the printing
     ! of the DATATRIEVE display.
     !
Print_Domain:
     CALL DTCMD (DAB, "PRINT !CMD;", DOMAIN)
     GOSUB Test_status

     ! Skip the subroutine.
     !
     GOTO Quit

     ! This subroutine prints messages, using DTMSG, and lines,
     ! using DTLINE.
Test_status:
     WHILE (DAB$W_STATE = DTR$K_STATE_MSG) OR &
           (DAB$W_STATE = DTR$K_STATE_LINE)
     SELECT DAB$W_STATE
     CASE DTR$K_STATE_MSG
             CALL DTMSG (DAB, MSGBUF, LENGTH)
             PRINT MSGBUF
             GOTO Quit IF DAB$W_ERR_SEV = SEV$K_SEVERE
     CASE DTR$K_STATE_LINE
             CALL DTLINE (DAB, MSGBUF, LENGTH)
             PRINT MSGBUF
     END SELECT
     CALL DTCONT(DAB)
     NEXT
     RETURN

     ! This call closes the interface.
     !
Quit:
     CALL DTFINI (DAB)
     END
```

4.2 DATATRIEVE States

After your program calls a DATATRIEVE–11 routine and the routine executes, DATATRIEVE enters a **state**, also called a **stallpoint**. That is, the routine places a value in the DAB$W_STATE field of the DAB and returns control to your program. At this point, DATATRIEVE is "stalling," waiting for another call from your program. The value of DAB$W_STATE determines what routine DATATRIEVE expects you to call next. If you call a routine that is not compatible with the current state, an error results.

For example, DATATRIEVE may be executing a statement, such as STORE, that normally prompts the user for input. It expects the user to pass a value. To indicate that it is waiting for a value, DATATRIEVE goes into the state DTR$K_STATE_PVAL; the routine places the value for DTR$K_STATE_PVAL (= 1) in the state field of the DAB. In this state, DATATRIEVE expects a call to the DTPVAL routine, which passes a value to DATATRIEVE. Similarly, when you pass a PRINT command to DATATRIEVE, DATATRIEVE goes into the state DTR$K_STATE_LINE, indicating that it expects a call to DTLINE next, to retrieve the print lines.

The following list briefly describes the states. Table 8–2 in Section 8.1 describes them in more detail.

DTR$K_STATE_INIT	The Call Interface has not been successfully initialized, or a call to DTFINI has closed the interface.
DTR$K_STATE_CMD	DATATRIEVE is waiting for a command or the continuation of a partial command. To continue, call DTCMD.
DTR$K_STATE_MSG	DATATRIEVE has a message ready to retrieve. To place the text of the message in the buffer you have specified, call DTMSG. To continue execution after retrieving the message, call DTCONT.
DTR$K_STATE_LINE	DATATRIEVE has executed a PRINT statement, but no device or file has been specified. To retrieve the print line and store it in a buffer, call DTLINE. To continue execution after displaying the line, call DTCONT. If a device or file has been specified, DATATRIEVE stores or displays the print lines there.
DTR$K_STATE_PVAL	DATATRIEVE has encountered a prompt and is waiting for the user to enter a value. To pass a value to DATATRIEVE, call DTPVAL.

DTR$K_STATE_GETP	DATATRIEVE has a record for the program to retrieve. To place the record in the buffer you have declared for it, call DTGETP. To continue execution after retrieving the record, call DTCONT.
DTR$K_STATE_PUTP	DATATRIEVE is waiting for your program to pass it a record. To pass a record to DATATRIEVE, call DTPUTP. To signal the end of a stream of records, call DTPEOF.

4.3 Declaring the DATATRIEVE Access Block (DAB)

The present DATATRIEVE state is not the only information DATATRIEVE stores in the DAB. DATATRIEVE also puts error severity codes and prompt strings in the DAB.

DATATRIEVE places the information in the DATATRIEVE Access Block for your program to read. You should think of the DAB as "read only." Your program should not modify the DAB. Table 4–1 shows the fields of the DAB in detail. For more information on each field of the DAB, see Section 8.1.

Table 4–1: The DATATRIEVE Access Block

Field	Length	Description
DAB$W_IDI	1 word	Internal identifier. You do not need to access this value.
DAB$W_STATE	1 word	The state of the DATATRIEVE–11 interface. When DATATRIEVE returns from a routine call, this field contains a value specifying the new state. Table 8–2 provides more information on DATATRIEVE states.
DAB$W_ERR_CODE	1 word	A 2-byte value associated with a DATATRIEVE message.
DAB$W_ERR_SEV	1 word	A value (0 to 4) representing the severity of the error listed in DAB$W_ERR_CODE.

(continued on next page)

Table 4–1 (Cont.): The DATATRIEVE Access Block

Field	Length	Description
DAB$W_FLAGS	1 word	Information passed from the DATATRIEVE routine to the calling program.
DAB$W_STR_LEN	1 word	The length of a string passed by DATATRIEVE to the calling program. This is the length of the string in DAB$V_STRING.
DAB$V_RESERVE	20 bytes	Not used. This area is reserved for future use.
DAB$V_STRING	n1 bytes	A string returned by a DATATRIEVE routine. This field contains a prompt string, port name, or other string. The length of this buffer is passed as the second parameter in the DTINIT call. DAB$W_STR_LEN contains the actual length of the string stored in DAB$V_ STRING. In the sample DAB inclusion files, the length of this field is 30 bytes.
DAB$V_BUFFER	n2 bytes	For internal use only. You should not access this field. The length of this field is passed as the third parameter to DTINIT. In the sample DAB inclusion files the length of this field is 150 bytes.

Your program must declare the fields of the DAB. However, you do not need to declare them explicitly for each program. Instead, you can create files containing the DAB definitions and include the file into each program with a single statement.

These files also declare other variables and constants that are often used in programs, such as error severity codes and initialization options. For example, you do not need to remember the required values for the str-len and buff-len parameters. These variables are declared and values assigned to them in the inclusion file. Simply use STRLEN and BUFLEN as the parameters to DTINIT.

There is one inclusion file for each of the supported high-level languages (COBOL, FORTRAN, and BASIC). An example of an inclusion file for each language appears in Appendix A. Simply include the appropriate file, using the INCLUDE command or its equivalent in your language.

To include the DAB in a FORTRAN program, use the following command:

```
INCLUDE 'DAB11.FTN'
```

In COBOL, the COPY command works the same way:

```
COPY "DAB11.CBL"
```

In BASIC-PLUS-2, use the INCLUDE directive:

```
%INCLUDE "DAB11.B2S"
```

If you are writing in a different language, you can create your own inclusion file to handle the DAB declarations. Model your file on the ones in Appendix A.

4.4 DATATRIEVE–11 Routines

Your program interacts with DATATRIEVE by calling external routines. There are 11 DATATRIEVE routines; each performs a different function. Table 4–2 briefly describes the function of each routine and the following sections describe how to use them in your programs.

Table 4–2: DATATRIEVE–11 Routines

DATATRIEVE Routine	Function
DTINIT	Initializes the DATATRIEVE interface.
DTCMD	Sends statements and commands for DATATRIEVE to execute.
DTLINE	After a DATATRIEVE statement (such as PRINT) that displays information executes, DTLINE retrieves the print line and stores it in a buffer. Call DTLINE for each line of text.
DTMSG	When DATATRIEVE has a message, DTMSG retrieves the message text and stores it in a buffer. Messages can have different severities, from success to severe error. See Chapter 8 for more information on severity codes.
DTCONT	After a call to DTLINE or DTMSG, DTCONT tells DATATRIEVE the message or print line has been received and the program is ready to continue to the next state.
DTPVAL	Sends a value to DATATRIEVE. Call DTPVAL in response to a DATATRIEVE prompt for a value. For example, after a STORE statement executes.
DTPUTP	Sends an entire record to DATATRIEVE. The record is passed to DATATRIEVE by way of a port. See Section 4.6 for more information on ports.
DTEOF	Sends an end-of-file mark to DATATRIEVE. When passing a record stream to DATATRIEVE with the DTPUTP routine, DTEOF indicates the end of the record stream.
DTGETP	Retrieves an entire record from DATATRIEVE. The record is passed to the program by way of a port. See Section 4.6 for more information on ports.
DTUNWD	Cancels a DATATRIEVE command. DTUNWD can be called from any DATATRIEVE state. It returns the interface to the state DTR$W_STATE_CMD.
DTFINI	Closes the DATATRIEVE interface.

4.4.1 Initializing the DATATRIEVE Interfaces

Setting up your program to use the DATATRIEVE–11 Call Interface involves two steps:

1. Declare the DATATRIEVE Access Block.
2. Initialize the interface.

After you include the DAB in your program, you initialize the DATATRIEVE Call Interface by calling the routine DTINIT. The syntax for DTINIT is as follows:

```
CALL DTINIT (dab, str-len, buff-len,<node>, options)
```

DTINIT sets up the DATATRIEVE Access Block, opens a path to the DECnet node on which DATATRIEVE will run, and specifies a set of DATATRIEVE options. See Chapter 8 for complete descriptions of the arguments.

The DAB inclusion file declares the DAB variable. It also declares the constants str-len and buff-len (using the names STRLEN and BUFLEN) and assigns values to them. If, for any reason, you need to change these constants, assign new values in the inclusion file.

The node parameter specifies the DECnet node name of the PDP–11 or VAX system on which the data is located. Specify this parameter on calls that use only the Remote Call Interface. The parameter must be blank if the call uses the Local Call Interface.

NOTE

Angle brackets surrounding *node* in the format mean that node is a string. In FORTRAN–77 programs, you need to specify two parameters for node: the string and its length. This is because FORTRAN passes parameters by address and length, rather than by descriptor, as in BASIC and COBOL.

Following is a typical FORTRAN call:

```
CALL DTINIT (DAB, STRLEN, BUFLEN, 'VACKS', 5, NOSEMI)
```

In BASIC, the same call is as follows:

```
CALL DTINIT (DAB, STRLEN, BUFLEN, "VACKS", NOSEMI)
```

In COBOL, declare variables and load them with values, either by including them in the program or by prompting the user for them:

```
01 NODE     PIC X(6).

DISPLAY "Enter node specification:  " WITH NO ADVANCING.
ACCEPT NODE.
CALL "DTINIT" DAB STRLEN BUFLEN
              BY DESCRIPTOR NODE
              BY REFERENCE NOSEMI.
```

4.4.2 Passing Commands to DATATRIEVE (DTCMD)

Once you have successfully initialized the interface with DTINIT, DATATRIEVE is at DTR$K_STATE_CMD. When DATATRIEVE is in this state, you can use DTCMD to pass an entire command, part of a command, or several commands to DATATRIEVE.

The format for DTCMD is as follows:

```
CALL DTCMD (dab, <command-str> [, <arg-str> ...])
```

See Chapter 8 for details.

In FORTRAN and BASIC, you can pass DATATRIEVE a literal command string as the command-str parameter. A typical FORTRAN call is as follows:

```
100      CALL DTCMD (DAB, 'READY YACHTS;' 13)
```

Notice that FORTRAN requires two parameters, the command and its length.

A BASIC call is as follows:

```
100     CALL DTCMD (DAB, 'READY YACHTS;')
```

Because BASIC and COBOL build a descriptor for a character string variable, you do not need the length parameter. However, in COBOL you must move the command to a string variable and pass the variable:

```
01   COMMAND PIC X(30) VALUE "READY YACHTS;".
          .
          .
          .
     CALL "DTCMD" USING DAB
                  BY DESCRIPTOR COMMAND.
```

Or:

```
     MOVE "PRINT YACHTS;" TO COMMAND.
     CALL "DTCMD" USING DAB
                  BY DESCRIPTOR COMMAND.
```

You can have your program prompt for a DATATRIEVE command and read the value for the command-str parameter from the terminal. For example, in BASIC-PLUS-2:

```
LINPUT "Enter a command to form a collection";COMMAND_LINE
CALL DTCMD (DAB, COMMAND_LINE)
```

You can also use the substitution directive !CMD with DTCMD. When you use !CMD as part of a DATATRIEVE command passed to DTCMD, DATATRIEVE replaces the directive with the string your program specifies.

For each substitution directive in the command string, you must include a parameter after the command string. The parameter must be a string descriptor (COBOL or BASIC) or the address and length of a string (FORTRAN). If your program uses substitution directives, you can change the values of parameters while the program is running. For example, in COBOL:

```
DISPLAY "Enter the domain to ready:  " WITH NO ADVANCING.
ACCEPT DOMAIN.
MOVE "READY !CMD;" TO COMMAND.
CALL "DTCMD" DAB BY DESCRIPTOR COMMAND DOMAIN.
```

The following FORTRAN code shows how to use DTCMD to pass a command line with a substitution directive to DATATRIEVE:

```
C
C Prompt for a domain.
C
100     WRITE (5,1000)
1000    FORMAT (' Enter the domain you wish to modify:  ', $)
C
C Read the user's input and its length.
C
        READ (5,2000)
2000    FORMAT (Q,A) LENGTH, COMAND
C
C Pass those values as parameters to DTCMD.
C
        CALL DTCMD (DAB, 'READY !CMD WRITE;', 17, COMAND, LENGTH)
```

You can also use DTCMD to construct long DATATRIEVE commands and statements. If you pass DATATRIEVE a fragment of a command or statement, the Call Interface is still at DTR$K_STATE_CMD after the routine executes. Then you can pass the continuation of the statement. DATATRIEVE does not execute the statement and change the state until it has received the entire statement.

```
CALL DTCMD (DAB, "STORE PT2 USING BEGIN;")
CALL DTCMD (DAB, "PART-A = TOTAL !CMD", FIELD1)
CALL DTCMD (DAB, "PART-B = TOTAL !CMD; END;", FIELD2)
!
! Call subroutine to check status.
!
GOSUB Test_status
```

For a more complete example, see the simple programs at the beginning this chapter. All the examples in this book contain calls to DTCMD.

4.5 Transferring Data

This section describes how your program and DATATRIEVE pass data back and forth. There are four types of data transfer through the Call Interface:

- Getting print lines from DATATRIEVE

  Interactive DATATRIEVE displays formatted text on the terminal or writes it to a file. Using DTLINE, your program can retrieve and display this text one line at a time.

- Getting messages from DATATRIEVE

  DATATRIEVE displays error messages and informational messages on the screen. Using DTMSG, your program can retrieve and display these messages.

- Passing values to DATATRIEVE

  Wherever interactive DATATRIEVE prompts for values, the Call Interface waits for user input. Your program uses DTPVAL to supply the input value in response to a prompt.

- Passing records to and retrieving records from DATATRIEVE

  To pass records between your program and DATATRIEVE, you use a port, which relates a structured record buffer declared in your program to a record defined in DATATRIEVE. To pass records to DATATRIEVE, your program calls DTPUTP; to retrieve records from DATATRIEVE, your program calls DTGETP. Section 4.6 explains how to use these routines in your program.

4.5.1 Retrieving Print Lines (DTLINE)

In interactive DATATRIEVE, a PRINT or Report Writer statement that does not specify a device or file causes the information to be displayed on the terminal. When you are calling DATATRIEVE from your program, you can retrieve the lines of a PRINT display using DTLINE.

The format for DTLINE is as follows:

```
CALL DTLINE (dab, <line-buf>, line-len)
```

Retrieving the lines of a DATATRIEVE display involves the following steps:

1. Declare a buffer to contain the print line. You can declare a single buffer in your program to contain print lines, error messages, and other text strings.

2. Declare a second variable to contain the length of the print line. DATATRIEVE stores in this variable the length of the print line that it places in your buffer.
3. Pass a PRINT statement using DTCMD. After the call to DTCMD, the DATATRIEVE state is DTR$K_STATE_LINE, indicating that DATATRIEVE has a print line to display.
4. Include a loop that calls DTLINE. The loop should do the following:
   a. Call DTLINE, passing as parameters the print line buffer and the length variable. DTLINE retrieves the line and places it in the buffer.
   b. Include a statement (such as WRITE, PRINT, or DISPLAY) to display the line, if you want to see it, or an assignment statement to store it.
   c. Call DTCONT to enter the next state.

   The program should loop until the state is no longer DTR$K_STATE_LINE, which indicates either that an error has occurred or that DATATRIEVE has finished printing lines.
5. Test the status code for success or error and take the required action.

An example of DTLINE appears in the following section as part of the general subroutine for handling messages and print lines.

4.5.2 Retrieving Messages (DTMSG)

When DATATRIEVE finishes executing a command or statement, it usually generates a message. This can be a success message (Statement completed successfully) or an error message (Element "BENEFITS" not found in dictionary). The procedure for displaying this message is similar to the procedure for displaying print lines. First, declare a message buffer in your program to contain this message. When you want to retrieve a message, call DTMSG using the buffer's address and length as parameters. DATATRIEVE places the message in the buffer. Your program can then display the message.

After a call to any DATATRIEVE routine, it is a good idea to test for errors and display any messages. You can use a subroutine to perform this function. An example of such a subroutine in each language appears at the end of this section.

The format for DTMSG is as follows:

```
CALL DTMSG (dab, <msg-buff>, msg-len)
```

For example, assume this COBOL call to DTMSG:

```
CALL DTMSG USING DAB
                 BY DESCRIPTOR MSGBUF
                 BY REFERENCE MSGLEN.
```

After the call to DTMSG, MSGBUF contains the message text and MSGLEN contains the length of the message. The program can then print the message and test its length.

At the same time, when DATATRIEVE enters the DTR$K_STATE_MSG state, it places a status code in one of the DAB fields. This field is named DAB$W_ERR_CODE in the BASIC DAB, DABERR in FORTRAN, and DAB-W-ERR-CODE in COBOL. The status is a binary number that identifies the specific error, if an error has occurred.

Finally, in the DTR$K_STATE_MSG state, DATATRIEVE places a severity code in another field of the DAB: DAB$W_ERR_SEV in BASIC, DABSEV in FORTRAN, and DAB-W-ERR-SEV in COBOL. Your program can test this value to determine whether a DATATRIEVE routine executed successfully and, if not, how severe the error was.

The steps for retrieving a message are similar to those for retrieving a print line:

1. Declare a buffer to contain the message. You can declare one buffer to contain both messages and print lines.
2. Declare a second variable to contain the length of the message. DATATRIEVE stores in this variable the length of the message line that it places in the message buffer.
3. Pass a command to DATATRIEVE using DTCMD. The execution of the command normally results in a message.
4. Include a loop that performs the following operations:
   a. Calls DTMSG, passing as parameters the message buffer and a variable to contain the message length
   b. Reads the error code and writes contents of the message buffer until all the messages have been displayed and the status code indicates success
   c. Calls DTCONT to continue to the next appropriate state

The following FORTRAN subroutine tests for and displays messages after a call to a DATATRIEVE routine. It also handles print lines. After most calls to DATATRIEVE routines, programs should call a subroutine that performs these functions. In this case, it displays both messages and print lines. The sample programs in this book all use a version of this subroutine to handle messages and print lines.

```
C
C This subroutine prints messages and print lines from DATATRIEVE.
C If the error is severe, it exits.  Otherwise, it returns the
C severity of the error to the main program.
C
        SUBROUTINE MESAGE  (SEV)
        INCLUDE 'DAB11.FTN'
        CHARACTER*80  MSGBUF
        INTEGER*2     MSGLEN
        INTEGER*4     SEV

1000    FORMAT (1X, A)

        SEV = SUCCES                                    (1)
10      IF (DABSTA .NE. DBSMSG) GO TO 20                (2)
        SEV = DABSEV                                    (3)
        CALL DTMSG (DAB, MSGBUF, 80, MSGLEN)            (4)
        WRITE (5,1000) MSGBUF                           (5)
        IF (SEV .EQ. SEVERE) GO TO 30                   (6)
        CALL DTCONT (DAB)                               (7)
        GO TO 10                                        (8)

20      IF (DABSTA .NE. DBSLIN) RETURN                  (9)
        CALL DTLINE (DAB, MSGBUF, 80, MSGLEN)           (10)
        WRITE (5,1000) MSGBUF)
        CALL DTCONT (DAB)
        GO TO 10

30      CALL DTFINI (DAB)                               (11)
        STOP 'SEVERE ERROR -- PROGRAM STOPPED'
        END
```

(1) SEV is a variable to contain the severity of the current error. The severity field in the DAB is defined only if the state is DBSMSG. Thus, if a call does not result in the state DBSMSG, the call was successful and the current contents of DABSEV do not apply to the call. Therefore, SEV must be initialized to the success value.

(2) If the current state is not "message," go on and see if it is "line."

(3) If the current state is "message," set the severity buffer to the severity of the current error.

(4) Call the routine to retrieve the message and place it in MSGBUF.

(5) Display the message.

(6) Exit if the error is severe.

(7) Continue to the next state.

(8) Go back and start again. There may be more messages.

(9) If the state is not "line," then there is neither a message nor a print line to display. Return to the program.

(10) If the state is "line," proceed as before: retrieve the line, print it, and continue.

⓫ Control reaches this line only if a severe error has been detected. The link to DATATRIEVE is disconnected and the program stops.

Following is a similar subroutine in BASIC:

```
Test_status:
        SEV = SEV$K_SUCCESS

        WHILE (DAB$W_STATE = DTR$K_STATE_MSG) OR &
              (DAB$W_STATE = DTR$K_STATE_LINE)

        SELECT DAB$W_STATE
                !
                ! If the state = MESSAGE, print message and check error.
                !
                CASE DTR$K_STATE_MSG
                        SEV = DAB$W_ERR_SEV
                        CALL DTMSG (DAB, MSGBUF, LENGTH)
                        PRINT MSGBUF
                        ! Quit if error is SEVERE
                        GOTO 8000 IF DAB$W_ERR_SEV = SEV$K_SEVERE
                !
                ! If the state = LINE, print the line.
                !
                CASE DTR$K_STATE_LINE
                        CALL DTLINE (DAB, MSGBUF, LENGTH)
                        PRINT MSGBUF
        END SELECT
        ! Continue to next state.
        CALL DTCONT(DAB)
        !
        ! Do this until all the lines and messages have been printed.
        !
        NEXT
        RETURN
```

The following COBOL paragraph performs the same functions:

```
900-PRINT-MESSAGES.
      IF DAB-W-STATE = DTR-K-STATE-MSG
          CALL "DTMSG" USING DAB
                  BY DESCRIPTOR MSGBUF
                  BY REFERENCE MSGLEN
          DISPLAY MSGBUF
          IF DAB-W-ERR-SEV = SEV-K-SEVERE GO TO 999-EOJ.
      IF DAB-W-STATE = DTR-K-STATE-LINE
          CALL "DTLINE" USING DAB
                  BY DESCRIPTOR MSGBUF
                  BY REFERENCE MSGLEN
          DISPLAY MSGBUF.
      CALL "DTCONT" USING DAB.
```

Execute this paragraph with the following statement:

```
PERFORM 900-PRINT-MESSAGES UNTIL
      DAB-W-STATE NOT = DTR-K-STATE-MSG AND
      DAB-W-STATE NOT = DTR-K-STATE-LINE.
```

In this way, COBOL tests for the correct state before each execution of the paragraph.

4.5.3 Passing Values to DATATRIEVE (DTPVAL)

When DATATRIEVE executes a statement, such as STORE, that displays a prompt and waits for the user's input, the Call Interface enters the state DTR$K_STATE_PVAL. At this point, your program must supply data in response to the prompt, either by passing a value itself or by reading a value from the terminal and passing it to DATATRIEVE. The program passes the value by calling DTPVAL.

When DATATRIEVE enters the state DTR$K_STATE_PVAL, it places the associated prompt in the DAB$V_STRING field of the DAB. You can retrieve this field and display it on the terminal to prompt for interactive input. In FORTRAN, there is one extra step. Because the DAB definition file declares the DAB$V_STRING field as a 30-byte array of the LOGICAL*1 data type, you must convert the field to a character string to display it on one line. To do this, declare an ASCII character string and use an EQUIVALENCE statement to map the LOGICAL*1 data to the CHARACTER data. See the example at the end of this section.

The format for DTPVAL is as follows:

```
CALL DTPVAL (dab, <value>)
```

The value parameter is the value that the user or the program has supplied in response to the prompt. The value passed to DATATRIEVE must be an ASCII character string. If the value is a real number or an integer, your program must convert it to a character string before passing it with DTPVAL.

To use this routine, do the following:

1. Pass the DATATRIEVE command or statement that issues the prompt.
2. If you want interactive input, retrieve the prompt from DAB$V_STRING and display the prompt. Then, include a language statement to read input from the terminal.

   If you want the program to define the data, include a language statement to place a value in the value parameter.
3. Call DTPVAL to send the value to DATATRIEVE.

The following FORTRAN program code illustrates how to modify values in a DATATRIEVE domain using DTPVAL:

```
          CHARACTER*30 PROMPT
          EQUIVALENCE (DABSTR, PROMPT)
                .
                .
                .
C Show the fields

100     CALL DTCMD (DAB, 'SHOW FIELDS;', 12)
        CALL MESAGE  (SEV)

C Prompt for a field value.
C
        WRITE (5,1000)
1000    FORMAT (' Which field do you wish to modify: ',$)
        READ (5,5000) LENGTH, FIELD
2000    FORMAT (Q,A)

C Call DTCMD with a modify command.  If the routine is successful,
C DATATRIEVE enters the DTR$K_STATE_PVAL state.
C
        CALL DTCMD (DAB, 'MODIFY !CMD;', 12, FIELD, LENGTH)
        CALL MESAGE  (SEV)

C If not successful, go back and try again or stop.
C
        IF (DABSTA .NE. DBSPMT) THEN
                CALL MESAGE  (SEV)
                WRITE (5,*) 'Try again or press CTRL/C to quit.'
                GOTO 100
        END IF

C If state is DTR$K_STATE_PVAL, retrieve the prompt from the string
C in the DAB, convert it to a string, and display it.  DABLEN is
C stored in the DAB.  It is the length of the prompt string.  This
C FORMAT statement displays only the prompt string, without trailing
C blanks.
C
200     WRITE (5,3000) PROMPT
3000    FORMAT (1X,A<DABLEN>,$)
        READ (5,2000) LENGTH, VALUE

C Call DTPVAL to pass the value to DATATRIEVE.

        CALL DTPVAL (DAB, VALUE, LENGTH)

        CALL MESAGE  (SEV)
300     RETURN
        END
```

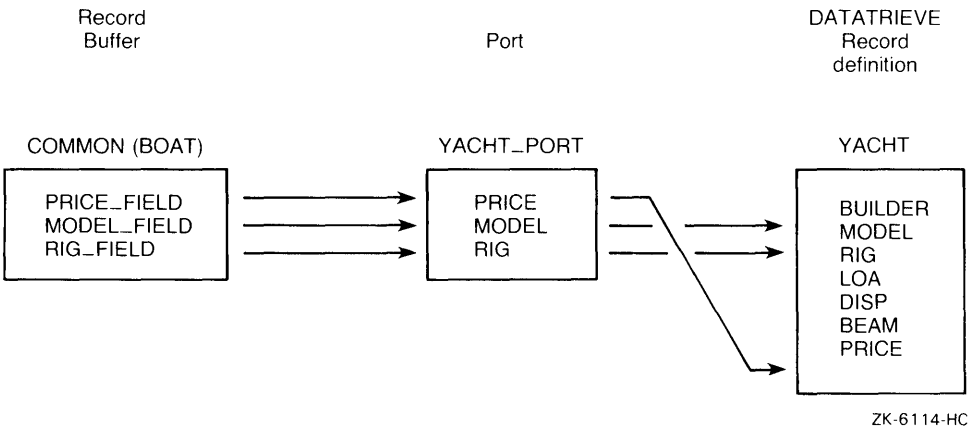
4.6 Transferring Records

It is often convenient to pass information between your program and DATATRIEVE in the form of records, rather than single values. This technique allows you greater flexibility in structuring your information. For example, when you use a STORE statement and DTPVAL to prompt for data and store it in a domain, the user must enter the values in the order they appear in the record definition. However, if you want your program to prompt for the values in a different order, you can prompt for input, place the input in the fields of a buffer you have declared, and pass the buffer as a record to DATATRIEVE.

You use **ports** to transfer records. A port is a kind of domain. It allows you to connect a DATATRIEVE domain, with its accompanying record structure, to a record buffer whose record structure you define in your program.

Ports also allow you to pass data types other than ASCII strings. The *value* parameter in DTPVAL must be a character string. With ports, however, you can build a record containing binary integers and floating point numbers as well, and pass the entire record to DATATRIEVE. Similarly, ports allow you to pass data of any data type from DATATRIEVE to your program.

Figure 4–1: The DATATRIEVE Port



4.6.1 Defining Ports

A port works like a domain. It is simply a name that ties together a record buffer declared in your program and a DATATRIEVE record definition.

To use a port, you must first declare a record structure for it in your program. In COBOL, for example, you might define a record buffer for YACHTS as follows:

```
01 BOAT.
        06 BUILDR PIC X(10).
        06 MODEL PIC X(10).
        06 RIG PIC X(6).
        06 LOA PIC 9(3).
        06 DISP PIC 9(5).
        06 BEAM PIC 9(2).
        06 PRICE PIC 9(5).
```

You define a port using a DEFINE PORT command or DECLARE PORT statement.

The DEFINE PORT command creates a port definition and places it in the DATATRIEVE data dictionary. You can issue the DEFINE PORT command either in interactive DATATRIEVE or with a DTCMD call from a program. Before using the port you must ready it. A port you create with DEFINE is still in effect after your program is finished. Thus, if the port is to be used for record transfer by more than one program, you might use DEFINE and make the port a permanent feature of the database.

The DECLARE PORT statement sets up a temporary port and readies it for write access. You must pass the DECLARE PORT statement with a DTCMD call. Using DECLARE has one major advantage. When you use DECLARE, the structure of the port is built into the program and always matches the record buffer declared there. If you define the port instead, someone else may change the definition and make it invalid for your program.

The following example defines a port using DEFINE, where YPORT is the name of the port and YACHT is the record definition from the YACHTS domain (defining the structure of YPORT):

```
DEFINE PORT YPORT USING YACHT;
```

The following COBOL example defines a port by passing a DECLARE command to DATATRIEVE. Notice that the fields are all declared as character data type. This is because the user will enter the data from the terminal.

```
MOVE "DECLARE PORT YPORT USING" TO COMMAND.
CALL "DTCMD" USING DAB BY DESCRIPTOR COMMAND.
MOVE "01 YACHT." TO COMMAND.
CALL "DTCMD" USING DAB BY DESCRIPTOR COMMAND.
MOVE "   06 BUILDER X(10)." TO COMMAND.
CALL "DTCMD" USING DAB BY DESCRIPTOR COMMAND.
MOVE "   06 MODEL X(10)." TO COMMAND.
CALL "DTCMD" USING DAB BY DESCRIPTOR COMMAND.
MOVE "   06 RIG X(6)." TO COMMAND.
CALL "DTCMD" USING DAB BY DESCRIPTOR COMMAND.
MOVE "   06 LOA XXX." TO COMMAND.
CALL "DTCMD" USING DAB BY DESCRIPTOR COMMAND.
MOVE "   06 DISP X(5)." TO COMMAND.
CALL "DTCMD" USING DAB BY DESCRIPTOR COMMAND.
MOVE "   06 BEAM XX." TO COMMAND.
CALL "DTCMD" USING DAB BY DESCRIPTOR COMMAND.
MOVE "   06 PRICE X(5).;" TO COMMAND.
CALL "DTCMD" USING DAB BY DESCRIPTOR COMMAND.
```

4.6.2 Retrieving Records from DATATRIEVE (DTGETP)

To transfer records from a DATATRIEVE domain to the record buffer you have declared in your program, you store the records in a DATATRIEVE port and call DTGETP to retrieve each record.

Use the STORE statement to store records into a port:

```
FOR YACHTS WITH LOA GT 30 STORE YPORT USING BEAM = BEAM
```

In the previous example:

`YACHTS WITH LOA GT 30`	Is a Record Selection Element (RSE) specifying the record stream.
`YPORT`	Is the name of a port where the records are stored.
`USING BEAM = BEAM`	Specifies a field from the records. In this case, only one field from the record is stored in the port. This name could also be a group field name, which would store multiple fields. If the USING clause contains the top-level record name (for instance, BOAT = BOAT), DATATRIEVE stores the entire record.

No records have been transferred at this point. The STORE statement simply associates the fields of the records in the data file with the appropriate fields in the record buffer that you have declared in your program. This tells DATATRIEVE what records you want it to store in the record buffer. Now the DATATRIEVE state is DTR$K_STATE_GETP. DATATRIEVE is waiting for a call to DTGETP to copy the fields of the data files into the fields of your record buffer.

Your program now calls DTGETP to get the record. The format for DTGETP is as follows:

```
CALL DTGETP (dab, <record-buf>, record-len)
```

In this call, record-buf is the record buffer you have declared in your program. DTGETP retrieves the record that was stored into the port and places it in record-buf. It also places the length of the record it has passed in record-len.

After DTGETP has executed, the state is still DTR$K_STATE_GETP. Call DTCONT to move to the next state. Next, you should test the state:

- If the state is DTR$K_STATE_GETP, there are still records in the record stream specified by the RSE in the STORE statement. Call DTGETP again to retrieve the next record.
- If the state is DTR$K_STATE_MSG, your program should test for success, display the contents of the message buffer, and continue.

The following BASIC program code shows how to retrieve a value from DATATRIEVE using DTGETP. In this case, the program has already created a port, PT1, containing the field N. In the following example, the program uses the field N to retrieve the count of the records in the current collection.

```
Find_collection:

        PRINT "Please enter a command to form a collection"
        LINPUT COMAND
        CALL DTCMD (DAB, COMAND)
        GOSUB Message

        COMAND = "STORE PT1 USING N = COUNT;"
        CALL DTCMD (DAB, COMAND)
        GOSUB Message)

        CALL DTGETP (DAB, COUNTERBUF, RECLEN)
        CALL DTCONT (DAB)
        GOSUB Message

        GOTO Find_collection IF COUNTERBUF = 0%
```

In the previous example:

- The STORE statement associates the value expression COUNT (the number of records in the current collection) with N, the first-level field name in the port PT1.
- DTGETP retrieves the value of N and places it in the variable COUNTERBUF. In the program LINEAR (see Section 7.2), COUNTERBUF is mapped to an integer variable, so that the count of records can be used in numeric calculations. After DTGETP executes, RECLEN, the length of the string in the buffer, is two.

- DTCONT returns control to the program and moves the Call Interface to the appropriate state. In this case, because there is only one value in the record stream, the state is probably DTR$K_STATE_MSG, indicating the success or failure of the routine.

4.6.3 Passing Records to DATATRIEVE (DTPUTP, DTPEOF)

You also use ports to pass records from your program to DATATRIEVE. Once you have declared the record buffer in your program and declared the port in DATATRIEVE, the transfer of a record is a 2-step process:

1. Use DTCMD to pass a statement that forms a record stream using the port. When DATATRIEVE detects a reference to the port, it enters the state DTR$K_STATE_PUTP.
2. Call DTPUTP to move the fields of the record from the record buffer to the port. DATATRIEVE maps the declared structure of the port to the record and executes the command passed in the previous step.

The format for DTPUTP is as follows:

```
CALL DTPUTP (dab, <record-buf>)
```

The record-buf parameter is the record buffer you have declared in your program.

When the last record has been passed, call DTPEOF:

```
CALL DTPEOF (dab)
```

This routine sends an end-of-file marker to DATATRIEVE.

The following FORTRAN example shows how to call DTPUTP and DTPEOF:

```
        INCLUDE 'DAB11.FTN'
        CHARACTER*80 MSGBUF
        CHARACTER*10 FIELD

        CALL DTINIT (DAB, STRLEN, BUFLEN, 'YRNODE', 6, 1)
        CALL MESAGE  (SEV)
        CALL DTCMD (DAB, 'SET DICTIONARY CDD$TOP.DTR$LIB.DEMO;', 36)
        CALL MESAGE  (SEV)

C Declare a port with one field of 10 characters.

        CALL DTCMD (DAB, 'DECLARE PORT TEST_PORT USING 01 TEST.', 37)
        CALL DTCMD (DAB, '   03 FIELD1 PIC X(10).;', 24)
        CALL MESAGE  (SEV)

C Pass a command to print the value in the port.

        CALL DTCMD (DAB, 'FOR TEST_PORT PRINT;', 20)
        CALL MESAGE  (SEV)
```

```
C There is no value in the port, so prompt for one.

150     WRITE (5,*) 'Enter a field.  Type CTRL/Z to quit.'
        WRITE (5,1700))
1700    FORMAT (' Field: ',$)
        READ (5, 1000, END = 200) FLDLEN, FIELD)
1000    FORMAT (Q,A)

C Now pass the value to DATATRIEVE.  At this point, the PRINT
C statement can be completed.  Loop back and prompt again.

        CALL DTPUTP (DAB, FIELD, 41)
        CALL MESAGE  (SEV)
        GO TO 150

C When the user types CTRL/Z (end-of-file), pass an end-of-file marker
C to DATATRIEVE.  This closes the record stream.

200     CALL DTPEOF (DAB)
        CALL MESAGE  (SEV)
        CALL DTFINI (DAB)
        WRITE (5,*) '         *****PROGRAM COMPLETED*****'
        END
```

This program does the following:

1. Sends a DECLARE PORT command to DATATRIEVE to set up a port. This port has only one 10-character field.
2. Passes a PRINT command that sets up a record stream whose source is the port. There are no values in the port yet, so DATATRIEVE enters DTR$K_STATE_PUTP, waiting for a value from the program.
3. Prompts the user to enter the value and set up CTRL/Z as the end-of-file marker.
4. Calls DTPUTP to pass that value to DATATRIEVE.
5. Executes the PRINT statement. The first time PRINT executes, DATATRIEVE adds the field name FIELD1 as a heading to the user's input.
6. Continues to prompt for and print the field value until the user types CTRL/Z. At this point, the program calls DTPEOF and exits.

The following FORTRAN program prompts the user for values for the YACHTS domain. These values are mapped to a record buffer using EQUIVALENCE statements. When the record is complete, the program calls DTPUTP to send the record to DATATRIEVE.

```
C
C Include the DAB definitions.
C Declare variables for the fields of the record.
C
        INCLUDE 'DAB11.FTN'
        INTEGER*4    SEV
        CHARACTER*1  ANSWER
        CHARACTER*20 DOMAIN
        CHARACTER*31 NODE, DICT
        CHARACTER*80 COMAND
        CHARACTER*41 YACHT
        CHARACTER*10 BILDER
        CHARACTER*10 MODEL
        CHARACTER*6  RIG
        CHARACTER*1  SPACE
        CHARACTER*2  LOA
        CHARACTER*5  DISP
        CHARACTER*2  BEAM
        CHARACTER*5  PRICE
        EQUIVALENCE  (YACHT(1:10),  BILDER)
        EQUIVALENCE  (YACHT(11:20), MODEL)
        EQUIVALENCE  (YACHT(21:26), RIG)
        EQUIVALENCE  (YACHT(27:27), SPACE)
        EQUIVALENCE  (YACHT(28:29), LOA)
        EQUIVALENCE  (YACHT(30:34), DISP)
        EQUIVALENCE  (YACHT(35:36), BEAM)
        EQUIVALENCE  (YACHT(37:41), PRICE)
        INTEGER*4    FLDLEN

        SPACE = ' '

C
C    Initialize the interface.
C
        WRITE (5,*) 'This program prompts for field values to be stored'
        WRITE (5,*) 'in the YACHTS domain.  It then puts the values'
        WRITE (5,*) 'into record form and passes them to DATATRIEVE.'
        WRITE (5,*) ' '

        WRITE (5,200)
200     FORMAT (' Node: ',$)
        READ (5,1000) LEN, NODE)
1000    FORMAT (Q,A)
        CALL DTINIT (DAB, STRLEN, BUFLEN, NODE, LEN, NOSEMI)
        CALL MESAGE  (SEV)

        WRITE (5,1100)
1100    FORMAT (' What dictionary do you want to use? ',$)
        READ (5,1000) LEN, DICT
        CALL DTCMD (DAB, 'SET DICTIONARY !CMD;', 20, DICT, LEN)
        CALL MESAGE  (SEV)
```

```
C
C    READY DOMAIN
C
100     WRITE (5,1200) 'Now we will ready the YACHTS domain.'
        CALL DTCMD (DAB, 'READY YACHTS WRITE;', 19)
        CALL MESAGE  (SEV)
        IF (SEV .EQ. ERROR) THEN
             WRITE (5,*) 'READY failed.  Try again'
             WRITE (5,*) 'or press CTRL/C to quit.'
             GO TO 100
        END IF

        CALL DTCMD (DAB, 'SHOW READY;', 11)
        CALL MESAGE  (SEV)

C
C Set up a port to pass records to DATATRIEVE.
C

        CALL DTCMD (DAB, 'DECLARE PORT BOAT_PORT USING 01 YACHT.', 38)
        CALL MESAGE  (SEV)
        CALL DTCMD (DAB, '03 BOAT.', 8)
        CALL MESAGE  (SEV)
        CALL DTCMD (DAB, '06 BUILDER PIC X(10).', 21)
        CALL MESAGE  (SEV)
        CALL DTCMD (DAB, '06 MODEL PIC X(10).', 19)
        CALL MESAGE  (SEV)
        CALL DTCMD (DAB, '06 RIG PIC X(6).', 16)
        CALL MESAGE  (SEV)
        CALL DTCMD (DAB, '06 LOA PIC X(3).', 16)
        CALL MESAGE  (SEV)
        CALL DTCMD (DAB, '06 DISP PIC X(5).', 17)
        CALL MESAGE  (SEV)
        CALL DTCMD (DAB, '06 BEAM PIC XX.', 15)
        CALL MESAGE  (SEV)
        CALL DTCMD (DAB, '06 PRICE PIC X(5).;', 19)
        CALL MESAGE  (SEV)

150     WRITE (5,*) '  Enter the fields in order.  '
        WRITE (5,*) '  Enter Control Z when through.'

        WRITE (5,1500)
1500    FORMAT (' Builder:  ',$)
        READ (5, 1000, END = 999) FLDLEN, BILDER

        WRITE (5,1600)
1600    FORMAT (' Model:  ',$)
        READ (5, 1000, END = 999) FLDLEN, MODEL

        WRITE (5,1700)
1700    FORMAT (' Rig: ',$)
        READ (5, 1000, END = 999) FLDLEN, RIG

        WRITE (5,1800)
1800    FORMAT (' Length: ',$)
        READ (5, 1000, END = 999) FLDLEN, LOA

        WRITE (5,1900)
1900    FORMAT (' Beam: ',$)
        READ (5, 1000, END = 999) FLDLEN, BEAM
```

```
        WRITE (5,2000)
2000    FORMAT (' Weight: ',$)
        READ (5, 1000, END = 999) FLDLEN, DISP

        WRITE (5,2100)
2100    FORMAT (' Price: ',$)
        READ (5, 1000, END = 999) FLDLEN, PRICE

        IF (DABSTA .EQ. DBSCMD) CALL DTCMD (DAB,
        1 'FOR BOAT_PORT STORE YACHTS USING BOAT = BOAT;', 45)
        CALL MESAGE  (SEV)

C
C Pass the complete record.
C

        CALL DTPUTP (DAB, YACHT, 41)
        CALL MESAGE  (SEV)

C
C    Make the user type "Y" to continue.
C

        WRITE (5,2200)
2200    FORMAT (' Do you wish to continue? [Y or N]  ', $)
        READ(5,3000) ANSWER
3000    FORMAT (A)
        IF ((ANSWER .EQ. 'Y') .OR. (ANSWER .EQ. 'y')) THEN
                GO TO 150
        END IF

C
C    Clean up and end the interface.
C

999     IF (DABSTA .EQ. DBSPPU) CALL DTPEOF (DAB)
        CALL MESAGE  (SEV)
        CALL DTFINI (DAB)

        WRITE (5,*) '          *****PROGRAM COMPLETED*****'
        END
```

4.7 Stopping the Execution of Commands

There are situations when you may want to stop DATATRIEVE from processing a command, discard the rest of the command, and return control to your program. For example, if the user presses CTRL/C while your program is prompting for a value for the DTPVAL routine, you do not want the program to exit. You want to cancel the STORE command that prompted for the value.

In such cases, you can use an error handler to trap CTRL/C and call the DTUNWD routine. This routine discards the remainder of the current command and returns DATATRIEVE to the state DTR$K_STATE_CMD.

The format for DTUNWD is as follows:

```
CALL DTUNWD (dab)
```

The sample BASIC program in Section 7.1 illustrates the use of DTUNWD in an error handling routine.

4.8 Closing the Call Interface

When your program is finished using the DATATRIEVE–11 Call Interface, it should call DTFINI. This routine acts like the EXIT command in interactive DATATRIEVE. It does cleanup operations such as releasing collections and variables, finishing domains, and closing files. It also breaks the link to the remote server on the node specified in DTINIT.

The format for DTFINI is as follows:

```
CALL DTFINI (dab)
```

The sample BASIC program in Section 7.1 illustrates the use of DTFINI.

Chapter 5

Sample FORTRAN Programs

This chapter contains several sample FORTRAN programs that call DATATRIEVE. These programs show how you can call DATATRIEVE to perform calculations on data, store and retrieve data, and create data management applications for end users.

5.1 Creating an End-User Interface

The program MENU shows you how to give users access to DATATRIEVE's data management capabilities. The program enables users unfamiliar with DATATRIEVE to display, store, modify, and report data managed by DATATRIEVE. The modules of the program also illustrate how to use the Call Interface to perform all the DATATRIEVE operations.

The main program MENU:

1. Initializes the interface
2. Chooses the DATATRIEVE dictionary
3. Opens a port PT1 to return the number of records in a collection
4. Calls the subroutine CHOOSE, which in turn:
   a. Displays the domains
   b. Readies the domain the user picks
   c. Returns
5. Displays a menu

Depending on the user's choice from the menu, MENU then calls one of seven subroutines:

ESTABLISH	Establishes a CURRENT collection.
DISPLAY	Displays the CURRENT collection.
SORT	Sorts the CURRENT collection.
MODIFY	Lets the user modify one record in the CURRENT collection, or one field for all the records in the CURRENT collection.
REPORT	Displays a report, based on the CURRENT collection, at the terminal.
STORE	Lets the user store new records in the readied domain.
CHOOSE	Lets the user ready a new domain.

In addition, some or all of these subroutines also call three other subroutines:

CLSCRN	Clears the terminal screen.
MESAGE	Tests for errors, messages, or print lines, and displays message and print lines. This subroutine appears in Section 4.5.2.
PROMPT	Prompts for a value and passes that value to DATATRIEVE.

For example, on an RSX–11M–PLUS system, the program uses a Task Builder command file such as the following:

```
MENU,MENU/-SP=MENU,MESAGE,CLSCRN,ESTABLISH,DISPLAY,
SORT,STORE,MODIFY,REPORT,CHOOSE,PROMPT,
LB:[1,1]F4POTS/LB,
LB:[1,1]RMSLIB/LB,
LB:[1,1]DTCLIB/LB:CIFOR:NC11M:NOLC,
LB:[1,1]DTCLIB/LB,
/
UNITS=10
GBLPAT=MENU:LUNMAP:177700:177777
//
```

Note that this task uses only the Remote Call Interface.

5.2 The Main Program: MENU

```
C Program:  MENU.FTN
C
C Include the DATATRIEVE Access Block.
C
        INCLUDE 'DAB11.FTN'

C Declare variables.

        CHARACTER*31 DOMAIN,NODE,DICT
        INTEGER*4    SEV
        INTEGER*4    LENGTH
        INTEGER*4    DOMLEN
        INTEGER*2    CHOICE

C Initialize the interface with DATATRIEVE.

5       WRITE (5,1000)
1000    FORMAT (' What node would you like to use?  ',$)
        READ (5,2000) LENGTH, NODE
2000    FORMAT (Q,A)
        CALL DTINIT (DAB, STRLEN, BUFLEN, NODE, LENGTH, NOSEMI)
        CALL MESAGE (SEV)
C
C Check for initialization error.  If initialization failed, go back
C and try again or quit.
C
        IF (DABSTA .EQ. DBSINI) THEN
                WRITE (5,*) 'Sorry, initialization failed on node ', NODE
                WRITE (5,1500)
1500    FORMAT (' Would you like to try another node? [Y or N] ',$)
                READ (5,3000) ANSWER
3000    FORMAT (A)
                IF ((ANSWER .EQ. 'Y') .OR. (ANSWER .EQ. 'y')) THEN
                        CALL DTFINI (DAB)
                        CALL MESAGE (SEV)
                        GO TO 5
                ELSE
                        GO TO 999
                END IF
        END IF

C   Clear the screen

10      CALL CLSCRN

C   Choose a dictionary:

        WRITE (5,3500)
3500    FORMAT (' What dictionary would you like to use?  ',$)
        READ (5,2000) LENGTH, DICT
        CALL DTCMD (DAB, 'SET DICTIONARY !CMD;', 20, DICT, LENGTH)
        CALL MESAGE (SEV)
```

```
        IF (SEV .EQ. ERROR) THEN
                WRITE (5,*) ' '
                WRITE (5,*) 'Sorry, try again.'
                WRITE (5,*) ' '
                GO TO 10
        END IF
C  Declare a port to contain the number of records in the domain to
C  be established.
        CALL DTCMD (DAB, 'DECLARE PORT PT1 USING ', 23)
        CALL DTCMD (DAB, '01 NUM PIC 9(4) USAGE IS COMP.;', 31)
        CALL MESAGE (SEV)

        IF (SEV .EQ. ERROR) THEN
                WRITE (5,*) 'Ports not declared.'
                WRITE (5,*) 'Program stopped.'
                GO TO 999
        END IF

C
C Call a subroutine to choose a domain.
C

20      CALL CHOOSE (DOMAIN, DOMLEN)

C
C The program displays a menu and prompts for a selection.
C

50      CHOICE = 0

        WRITE (5,60)
60      FORMAT (//23X,'MENU'/23X' '/
        1/10X,'1. Establish a collection of records.'
        2/10X,'2. Display the current collection.'
        3/10X,'3. Sort the current collection.'
        4/10X,'4. Update the current collection.'
        5/10X,'5. Report the current collection.'
        6/10X,'6. Store records in the current domain.'
        7/10X,'7. Choose another domain.'
        8/10X,'8. End this session.'/////
        1/10X'    Enter the number of the operation '
        2/10X'    you wish to perform: ',$)

        READ (5,70)CHOICE
70      FORMAT (I2)

C Call the appropriate subroutine to handle the choice entered.

        IF ((CHOICE .LT. 1) .OR. (CHOICE .GT. 8)) THEN
                WRITE (5,*) 'Please enter a number from 1 to 8.'
                GO TO 50
        END IF
        IF (CHOICE .EQ. 1) CALL ESTABL (DOMAIN, DOMLEN)
        IF (CHOICE .EQ. 2) CALL DISPLA
        IF (CHOICE .EQ. 3) CALL SORT (DOMAIN, DOMLEN)
        IF (CHOICE .EQ. 4) CALL MODIFY (DOMAIN, DOMLEN)
        IF (CHOICE .EQ. 5) CALL REPORT (DOMAIN, DOMLEN)
        IF (CHOICE .EQ. 6) CALL STORE (DOMAIN, DOMLEN)
```

```
C First finish current domain, then erase the screen and call the
C subroutine to choose a domain.

        IF (CHOICE .EQ. 7) THEN
                CALL DTCMD (DAB, 'FINISH !CMD;', 12, DOMAIN, DOMLEN)
                CALL MESAGE (SEV)
                CALL CLSCRN
                GO TO 20
        END IF

C Finish the session with DATATRIEVE and stop the program.

        IF (CHOICE .EQ. 8) GO TO 999

C Return to the menu.

        GO TO 50

999     CALL DTFINI (DAB)
        END
```

5.3 The ESTABLISH Subroutine

```
C*********************************************************************
C                    SUBROUTINE ESTABLISH                             *
C The program searches the current domain for records that fit the    *
C description given to DATATRIEVE and forms a collection.             *
C*********************************************************************

        SUBROUTINE ESTABL (DOMAIN, DOMLEN)
        INCLUDE 'DAB11.FTN'
        CHARACTER*1  ANSWER
        CHARACTER*31 ATTR
        CHARACTER*31 BOOL
        CHARACTER*31 DOMAIN
        CHARACTER*31 VALUE
        CHARACTER*31 PORLEN
        INTEGER*2    NUMREC
        INTEGER*4    DOMLEN
        INTEGER*4    SEV
        INTEGER*4    LEN
        DIMENSION    LEN(5)

C Call a subroutine to clear the screen.

        CALL CLSCRN

C Ask if the user wishes to use all the records in the domain.
C If the response is yes, issue the FIND command for the whole domain.

100     WRITE (5,1000)
1000    FORMAT (' Do you wish to use all the records
        1 in the domain? [Y or N]  ',$)
        READ (5,2000) ANSWER
```

```
2000    FORMAT (A)
        IF (ANSWER .EQ. 'Y' .OR. ANSWER .EQ. 'y')  THEN
                CALL DTCMD (DAB, 'FIND !CMD;', 10, DOMAIN, DOMLEN)
                CALL MESAGE (SEV)
                RETURN
        ELSE
                IF (ANSWER .NE. 'N' .AND. ANSWER .NE. 'n') THEN
                        WRITE (5,*) 'Please enter YES or NO.'
                        GO TO 100
                END IF
        END IF

C Show the user the fields available for a record selection
C expression.

105     CALL DTCMD (DAB, 'SHOW FIELDS;', 12)
        CALL MESAGE (SEV)

C Prompt the user for the different parts of a record selection
C expression, a field name, a relational operator, and a value.

        WRITE (5,3000)
3000    FORMAT (/10X, ' The collection will be formed on the basis of'
        1 /10X, ' identifying characteristics you choose. '
        2 /10X, ' Specify these characteristics by entering'
        3 /10X, ' a FIELD, a RELATION, and a VALUE. For'
        4 /10X, ' example, if your domain is EMPLOYEES, you can form'
        5 /10X, ' a collection of:'
        6 //15X, ' EMPLOYEES whose SALARY (FIELD) is'
        7 /15X, ' GT (RELATION) $25,000 (VALUE).'
        8 //' Enter the FIELD (SALARY, PRICE, DEPARTMENT):
        READ (5,3500) LEN(2), ATTR
3500    FORMAT (Q,A)

        WRITE (5,4000)
4000    FORMAT (' Enter the RELATION (EQ, GT, GE, LT, CONTAINING): ',$)
        READ (5,3500) LEN(3), BOOL

        WRITE (5,5000)
5000    FORMAT (' Enter the VALUE (non-numeric values should be in quotes):
        1 ',$)
        READ (5,3500) LEN(4), VALUE

C Instruct DATATRIEVE to find the desired records.

        CALL DTCMD
        1  (DAB, 'FIND CURRENT WITH !CMD !CMD !CMD;', 33, ATTR, LEN(2),
        2 BOOL, LEN(3), VALUE, LEN(4))

        CALL MESAGE (SEV)

C Verify that the FIND was completed successfully.

        IF (SEV .EQ. ERROR) THEN
                WRITE (5,*) 'Sorry, collection not established.'
                WRITE (5,*) 'Please try again.'
                GO TO 105
        END IF
```

```
C Verify that there were records found.
C If no records were found, the user must either use all
C records in the collection or establish a new collection.

        CALL DTCMD (DAB, 'STORE PT1 USING NUM = COUNT;', 28)
        CALL MESAGE (SEV)

C If state is DTR$K_STATE_GETP then issue call to DTGETP
C to retrieve the number of records found.

        IF (DABSTA .EQ. DBSPGE) THEN
                CALL DTGETP (DAB, NUMREC, 2, PORLEN)
                CALL DTCONT (DAB)
                CALL MESAGE (SEV)
        END IF

C If no records were found, notify the user and find all the
C records in the domain.  This prevents a collection with no
C records.  Prompt the user to continue or return to the main menu.

        IF (NUMREC .EQ. 0) THEN
                CALL DTCMD (DAB, 'FIND !CMD;', 10, DOMAIN, DOMLEN)
                CALL MESAGE (SEV)
                WRITE (5,*) ' There are no records that fit.'
                GO TO 100
        END IF
        CALL MESAGE (SEV)

C Ask if the user wishes to make a subcollection.  If not, return.

160     WRITE (5,7000)
7000    FORMAT (' Would you like to establish a sub-collection'
        1/' from the current collection?  [Y or N]  ',$)
        READ (5,2000) ANSWER
        IF (ANSWER .EQ. 'Y' .OR. ANSWER .EQ. 'y') THEN
                GO TO 105
        END IF

        RETURN
        END
```

5.4 The DISPLAY Subroutine

```
C***************************************************************
C                      DISPLAY                                  *
C This subroutine displays the current collection of records.   *
C***************************************************************

        SUBROUTINE DISPLA
        INCLUDE 'DAB11.FTN'
        INTEGER*4    SEV
        CHARACTER*1  CR

        CALL CLSCRN

C
C Have DATATRIEVE print the current collection.
C
        CALL DTCMD (DAB, 'PRINT CURRENT;', 14)
        CALL MESAGE (SEV)

C
C Put a message at the bottom of the page.
C

        WRITE (5,1000)
1000    FORMAT (' Press RETURN to continue > ',$)

C
C When the user types a character, return.
C

        READ (5,1) CR
1       FORMAT (A)
        RETURN
        END
```

5.5 The SORT Subroutine

```
C*********************************************************************
C                              SORT                                   *
C This subroutine sorts the current file in ascending or descending   *
C order.                                                              *
C*********************************************************************

        SUBROUTINE SORT (DOMAIN, DOMLEN)
        INCLUDE 'DAB11.FTN'
        CHARACTER*1  CR
        CHARACTER*1  ORD
        CHARACTER*27 TEXT
        CHARACTER*31 FIELDS
        INTEGER*4    FLEN, SEV
        LOGICAL      UNSORT, NOORD

        UNSORT = .TRUE.
        NOORD = .TRUE.

C Set up FORMAT statements.
```

```
1000    FORMAT (' Press RETURN to continue >',$)
2000    FORMAT (Q,A)
3000    FORMAT (A)

C Display the available fields.

300     CALL DTCMD (DAB, 'SHOW FIELDS;', 12)
        CALL MESAGE (SEV)
        WRITE (5,4000)
4000    FORMAT (' Enter the FIELD by which you wish to sort: ',$)
        READ (5,2000) FLEN, FIELDS

C Continue in loop until a correct sorting order has been entered.

320     WRITE (5,5000)
5000    FORMAT (' Sort in ascending or descending order (A or D)? ',$)

C Prompt for sort order, then issue a DATATRIEVE command to sort
C the current collection by the field chosen and in the order chosen.

        READ (5,3000) ORD
        IF ((ORD .EQ. 'A') .OR.
        1 (ORD .EQ. 'a'))  THEN

C Sort by the ascending field given by the user.

                NOORD = .FALSE.
                CALL DTCMD (DAB, 'SORT CURRENT BY ASCENDING !CMD;',
        1          31, FIELDS, FLEN)
        ELSE
                IF ((ORD .EQ. 'D') .OR.
        1         (ORD .EQ. 'd'))  THEN

C Sort by the descending field given by the user.

                        NOORD = .FALSE.
                        CALL DTCMD (DAB, 'SORT CURRENT BY DESCENDING !CMD;',
        1             32, FIELDS, FLEN)
                ELSE
                        WRITE (5,6000)
6000    FORMAT (' Re-enter sorting order')
                END IF
        END IF

        IF (NOORD) GO TO 320

        NOORD = .TRUE.
        CALL MESAGE (SEV)

        IF (SEV .EQ. ERROR) THEN
                WRITE (5,1000)
                READ (5,3000) CR
        ELSE
                UNSORT = .FALSE.
        END IF

        IF (UNSORT) THEN
                CALL CLSCRN
                GO TO 300
        END IF

        UNSORT = .TRUE.

C Inform user that sort is complete.
```

```
370     WRITE (5,7000)
7000    FORMAT (/////26X,'Sort successfully completed.')
        WRITE (5,1000)
        READ (5,3000) CR
        RETURN
        END
```

5.6 The MODIFY Subroutine

```
C***************************************************************
C                       MODIFY                                  *
C This subroutine sets up a second menu and modifies records.   *
C***************************************************************

        SUBROUTINE MODIFY (DOMAIN, DOMLEN)
        INCLUDE 'DAB11.FTN'
        CHARACTER*1  CR
        CHARACTER*1  ANSWER
        CHARACTER*31 DOMAIN, FIELD, VALUE
        CHARACTER*80 CHLINE, MSGBUF
        INTEGER*4    NUMBER, LENGTH, DOMLEN, SEV
        INTEGER*2    CHOICE, NUM

        CHOICE = 1

C Erase the screen.

400     CALL CLSCRN

C Display the MODIFY submenu.

1250    FORMAT (' Press RETURN to continue >',$)

        WRITE (5,1500)
1500    FORMAT (////'    1. One or more fields for one record.'/'
        1    2. One field for all records in the current collection.'/'
        2    3. Return to main menu'///' Enter your choice: ',$)
        READ (5,2000) CHOICE
2000    FORMAT (I1)

C Issue a command to DATATRIEVE to start with the first record in the
C current collection.

        CALL DTCMD (DAB, 'SELECT 1;', 9)
        CALL MESAGE (SEV)
        GO TO (410, 430, 499), CHOICE
        WRITE (5,2500)
2500    FORMAT (' Invalid operation...try again.')
        WRITE (5,1250)
        READ (5,3000) CR
3000    FORMAT(A)
        GO TO 400

C Select records one at a time. Prompt for the record(s) the user
C wishes to modify.

409     CALL DTCMD (DAB, 'SELECT NEXT;', 12)
        CALL MESAGE (SEV)

410     CALL CLSCRN
```

```
C Display the selected record.

411     CALL DTCMD (DAB, 'PRINT;', 6)
        CALL MESAGE (SEV)

C Inquire if this record needs modification.

        WRITE (5,3500)
3500    FORMAT (//' Is this the record you wish to update?'//
        1 '  Enter YES, NO, or EXIT : ',$)
        READ (5,3000) ANSWER
        IF (ANSWER .EQ. 'E' .OR. ANSWER .EQ. 'e') GO TO 400
        IF (ANSWER .NE. 'Y' .AND. ANSWER .NE. 'y') THEN
                NUMBER = NUMBER + 1
                GO TO 409
        END IF
        CALL CLSCRN

C Show the fields.

413     CALL DTCMD (DAB, 'SHOW FIELDS;', 12)
        CALL MESAGE (SEV)
        WRITE (5,4500)
4500    FORMAT (' Which field do you wish to modify: ',$)
        READ (5,1000) LENGTH, FIELD
1000    FORMAT (Q,A)

C Modify and check for errors.

        CALL DTCMD (DAB, 'MODIFY !CMD;', 12, FIELD, LENGTH)
        CALL MESAGE (SEV)

        IF (DABSTA .NE. DBSPMT) THEN
                WRITE (5,5500)
5500            FORMAT(' Do you want to try again? ',$)
                READ (5,3000) ANSWER
                IF ((ANSWER .EQ. 'N') .OR. (ANSWER .EQ. 'n')) THEN
                    GO TO 400
                ELSE
                    GO TO 411
                END IF
        END IF

        CALL PROMPT

        IF (SEV .EQ. ERROR) THEN
                WRITE (5,*) 'Try again.'
                GO TO 411
        END IF

        CALL CLSCRN

C Print the modified record.

        CALL DTCMD (DAB, 'PRINT;', 6)
        CALL MESAGE (SEV)

427     WRITE (5,6000)
6000    FORMAT (/' Do you wish to modify any more fields in this record? ',$)

        READ (5,3000) ANSWER
        IF (ANSWER .EQ. 'Y' .OR. ANSWER .EQ. 'y') GO TO 413
        WRITE(5,6500)
```

```
6500    FORMAT (/' Do you wish to continue updating records? ',$)
        READ (5,3000) ANSWER
        IF (ANSWER .EQ. 'Y' .OR. ANSWER .EQ. 'y') THEN
                NUMBER = NUMBER +1
                GO TO 409
        END IF
        GO TO 400

C Modify one field for all the records in the current collection.

430     CALL CLSCRN
432     WRITE (5,7000)
7000    FORMAT (/////' Do you want to:'//'
        1     1. Update all records using one value.'/'
        2     2. Update all records with an equation '/'
        3        (for example, price = price + 400)'/'
        4     3. Return to the previous menu. '/'
        5     Enter 1, 2 or 3: '$)
        READ (5,2000) NUM
        GO TO (460,445,400), NUM

C Prompt for an equation.

445     CALL CLSCRN

C Show the fields.

        CALL DTCMD (DAB, 'SHOW FIELDS;', 12)
        CALL MESAGE (SEV)
        WRITE (5,7500)
7500    FORMAT (' Enter the equation you wish to use: ',$)
        READ (5,1000) LENGTH, CHLINE
        CALL DTCMD (DAB, 'MODIFY ALL USING !CMD;', 22, CHLINE, LENGTH)
        CALL MESAGE (SEV)

C If not successful, show the fields and start again.

        IF (SEV .EQ. ERROR) THEN
                GO TO 430
        END IF

        CALL CLSCRN
        WRITE (5,8000)
8000    FORMAT (' All records updated.')

C Print the updated collection.

        CALL DTCMD (DAB, 'PRINT CURRENT;', 14)
        CALL MESAGE (SEV)
        WRITE (5,1250)
        READ (5,3000) CR
        GO TO 400

C Modify one field for all records.

460     CALL CLSCRN
        CALL DTCMD (DAB, 'SHOW FIELDS;', 12)
        CALL MESAGE (SEV)
        WRITE (5,8500)
8500    FORMAT (' Which field do you wish to update? ',$)
        READ (5,1000) LENGTH, FIELD

C Issue the modify command to DATATRIEVE.
```

```
        CALL DTCMD (DAB, 'MODIFY ALL !CMD OF CURRENT;', 27, FIELD, LENGTH)
        CALL MESAGE (SEV)

C Check to see if it was successful.

        IF (SEV .EQ. ERROR) THEN
                WRITE (5,*) 'Try again.'
                GO TO 430
        END IF

C Prompt for a value.

        CALL PROMPT

        IF (SEV .EQ. ERROR) THEN
                WRITE (5,9500)
9500            FORMAT (' Invalid, try again....'//)
                GO TO 430
        END IF

        CALL CLSCRN

C Print the modified collection.

        CALL DTCMD (DAB, 'PRINT CURRENT;', 14)
        CALL MESAGE (SEV)
        WRITE (5,1250)
        READ (5,3000) CR

        IF (CHOICE .NE. 3) GO TO 400

499     RETURN
        END
```

5.7 The REPORT Subroutine

```
C*********************************************************************
C                          REPORT                                     *
C The subroutine invokes the DATATRIEVE Report Writer and prompts     *
C for the information necessary to write the report. The user can report *
C the whole file or a specific collection.                            *
C*********************************************************************

        SUBROUTINE REPORT (DOMAIN, DOMLEN)
        INCLUDE 'DAB11.FTN'
        CHARACTER*1  ANSWER,CR
        CHARACTER*80 RPTHDR, CHLINE, MSGBUF
        CHARACTER*75 SHOBUF(100)
        CHARACTER*31 DOMAIN
        INTEGER*4    LENGTH, DOMLEN, SEV

C Store the output from a "SHOW FIELDS" command in an array to be
C displayed later, when the user must choose field names.
```

```
        I = 0
        CALL DTCMD (DAB, 'SHOW FIELDS;', 12)
        SEV = SUCCES
500     IF (DABSTA .EQ. DBSMSG) THEN
                SEV = DABSEV
                IF (SEV .EQ. SEVERE) CALL MESAGE (SEV)
                IF (SEV .EQ. ERROR) GO TO 550
                CALL DTMSG (DAB, MSGBUF, 80, LEN)
                WRITE (5,*) MSGBUF
                SHOBUF(I) = MSGBUF
                I = I + 1
                CALL DTCONT (DAB)
                GO TO 500
        END IF

C Ask if the user wants to use the whole domain or a collection.

505     CALL CLSCRN
        WRITE (5,1000)
1000    FORMAT (///' Do you wish to limit the types of records in
        1 the report? '/' (For example, records with PRICE GT 20000'/
        2' or records with DEPARTMENT EQ "SERVICE" SORTED BY EMPLOYEE_NUMBER)
        4 /' Enter YES or NO: ',$)
        READ (5,2000) ANSWER
2000    FORMAT(A)

C Ask the user if he or she wants a record selection expression on
C the report.

        IF ((ANSWER .EQ. 'Y') .OR. (ANSWER .EQ. 'y')) THEN
                CALL DTCMD (DAB, 'SHOW FIELDS;', 12)
                CALL MESAGE (SEV)

C Prompt for the RSE and pass it to DATATRIEVE.

                WRITE (5,3000)
3000    FORMAT (///' Enter an expression such as PRICE GT 2000 or'/
        1 ' DATE_EMPLOYED AFTER "01-JULY-1980")'///'Expression: ',$)
                READ (5,4000) LENGTH, CHLINE
4000    FORMAT (Q,A)
                CALL DTCMD (DAB, 'REPORT CURRENT WITH !CMD;', 25,
        1 CHLINE, LENGTH)
        ELSE

C Invoke the report writer for the whole file.

                CALL DTCMD (DAB, 'REPORT CURRENT;', 15)
        END IF

C Check for errors.

        CALL MESAGE (SEV)
        IF (SEV .EQ. ERROR) GO TO 550

C Prompt for a report title.

525     WRITE (5,5000)
5000    FORMAT (///' Enter the report title enclosed in quotation marks'
        1/' Separate lines with a slash (/)'/
        2' For example, enter:'/'     "RATES SCHEDULE"/"DOMESTIC"'/
        3' to produce the title:     RATES SCHEDULE'/
        4'                              DOMESTIC')
        READ (5,4000) LENGTH, RPTHDR
```

```
C Set the report title.

        IF (LENGTH .EQ. 0) THEN
           CALL DTCMD (DAB, 'SET REPORT_NAME = "";', 21)
        ELSE
           CALL DTCMD (DAB, 'SET REPORT_NAME = !CMD;', 23, RPTHDR, LENGTH)
        END IF

        CALL MESAGE (SEV)
        IF (SEV .EQ. ERROR) GO TO 550

C Set more characteristics of the report.

        CALL DTCMD (DAB, 'SET LINES_PAGE = 22;', 20)
        CALL MESAGE (SEV)

C Show the user the previously stored fields.

        DO 538 J = 1, I
                WRITE (5,6000) SHOBUF(J)
6000    FORMAT (1X,A75)
538     CONTINUE

C Prompt the user for field names.

        WRITE (5,7000)
7000    FORMAT (' Enter the fields you wish to have in the report.
        1 Separate them by commas. ', $)
        READ (5,4000) LENGTH, CHLINE

C Pass the field names to DATATRIEVE.

        CALL DTCMD (DAB, 'PRINT !CMD;', 11, CHLINE, LENGTH)
        CALL DTCMD (DAB, 'END_REPORT;', 11)
        CALL MESAGE (SEV)

        WRITE (5,7500)
7500    FORMAT (' Press RETURN to continue > ',$)
        READ (5,7600) CR
7600    FORMAT (A)
        CALL CLSCRN

C If not successful, prompt the user to start over.

        IF (SEV .NE. ERROR) RETURN

550     WRITE (5,8000)
8000    FORMAT (' An error was found by the Report Writer,'/'
        1 >>>>>Do you want to try again? ',$)
        READ (5,2000) ANSWER
        IF ((ANSWER .EQ. 'Y') .OR. (ANSWER .EQ. 'y')) GO TO 505

        RETURN
        END
```

5.8 The STORE Subroutine

```
C*********************************************************************
C                          STORE                                      *
C The subroutine allows the user to store records in the current      *
C domain.                                                             *
C*********************************************************************

        SUBROUTINE STORE (DOMAIN, DOMLEN)
        INCLUDE 'DAB11.FTN'
        CHARACTER*10 NUMBER
        CHARACTER*31 DOMAIN
        INTEGER*4    LENGTH
        INTEGER*4    DOMLEN
        INTEGER*4    SEV

        CALL CLSCRN

C Prompt the user for the number of records to be stored.  This way,
C only one DTCMD call has to be made to store multiple records.

100     WRITE (5,1000)
1000    FORMAT (' Enter the number of records you wish to store: ',$)
        READ (5,2000) LENGTH, NUMBER
2000    FORMAT (Q,A)

        CALL CLSCRN

        CALL DTCMD (DAB,'REPEAT !CMD STORE !CMD;', 23,
        1 NUMBER, LENGTH, DOMAIN, DOMLEN)
        CALL MESAGE (SEV)

        CALL PROMPT

        IF (SEV .EQ. ERROR) THEN
                WRITE (5,*) 'Last record not stored.  Try again.'
                GO TO 100
        END IF

C Issue a command to find all of the records, so the newly stored
C records are in the current collection.

200     CALL DTCMD (DAB, 'FIND !CMD;', 10, DOMAIN, DOMLEN)
        CALL MESAGE (SEV)
        RETURN
        END
```

5.9 The CHOOSE Subroutine

```
C*********************************************************************
C                            CHOOSE                                   *
C The subroutine shows the domains available in the current dictionary *
C and prompts the user to ready a domain before entering the program.  *
C If the domain name is invalid or the domain cannot be readied, the   *
C program reprompts for a domain name.                                 *
C*********************************************************************

        SUBROUTINE CHOOSE (DOMAIN, DOMLEN)
        INCLUDE 'DAB11.FTN'
        CHARACTER*31 DOMAIN
        INTEGER*4    DOMLEN
        INTEGER*4    SEV
        LOGICAL      NODOM

        NODOM = .TRUE.

10      CALL DTCMD (DAB, 'SHOW DOMAINS;', 13)
        CALL MESAGE (SEV)

C Ask the user for the domain and ready it.

        WRITE (5,2000)
2000    FORMAT (' Enter the name of the domain you want to use: ' ,$)
        READ (5,1000) DOMLEN, DOMAIN
1000    FORMAT (Q,A)
        CALL DTCMD (DAB, 'READY !CMD WRITE;', 17, DOMAIN, DOMLEN)

C Check for an error in readying the domain.  Prompt again if an error
C occurred.  Then form a collection of all records in the domain and
C check for errors.

        CALL MESAGE (SEV)

        IF (SEV .EQ. ERROR) THEN
                WRITE (5,3000)
3000    FORMAT (' Try again....')
        ELSE
                NODOM = .FALSE.
                CALL DTCMD (DAB, 'FIND !CMD;', 10, DOMAIN, DOMLEN)
                CALL MESAGE (SEV)
                IF (SEV .EQ. ERROR) THEN
                        WRITE (5,3000)
                        NODOM = .TRUE.
                END IF
        END IF
        IF (NODOM) GO TO 10
        NODOM = .TRUE.
        RETURN
        END
```

5.10 The PROMPT Subroutine

```
C ***************************************************************
C *                          PROMPT                             *
C * This subroutine displays a DATATRIEVE prompt, and sends a   *
C * value to the DTPVAL routine.                                *
C ***************************************************************

        INCLUDE 'DAB11.FTN'
        CHARACTER*20 VALUE
        INTEGER*4    LENGTH, SEV
        CHARACTER*30 PR
        EQUIVALENCE  (PR, DABSTR)

100     IF (DABSTA .EQ. DBSPMT) THEN
                WRITE (5,1000) PR
1000            FORMAT (X,A<DABLEN>,$)
                READ (5,2000) LENGTH, VALUE
2000            FORMAT(Q,A)
                CALL DTPVAL (DAB, VALUE, LENGTH)
                CALL MESAGE (SEV)
                IF (SEV .EQ. ERROR) GO TO 200
                GO TO 100
        END IF

200     RETURN
        END
```

5.11 The CLSCRN Subroutine

```
C ***************************************************************
C *                          CLSCRN                             *
C * This subroutine clears the screen and moves the cursor to   *
C * home by issuing the appropriate  VT-100 escape sequences.   *
C ***************************************************************

        BYTE ESC
        ESC = 155

        WRITE (5,10) ESC
10      FORMAT (X,1A1,'[2J')
        WRITE (5,20) ESC
20      FORMAT (X,1A1,'[H')
        END
```

Chapter 6

Sample COBOL Programs

This chapter contains two sample COBOL programs that call DATATRIEVE. These programs show how you can call DATATRIEVE to perform calculations on data, to store and retrieve data, and to help end users perform information management tasks.

6.1 Creating an End-User Interface

The program ENTRY accepts data entered by a user from the terminal and stores the data in the DATATRIEVE domain YACHTS. The program uses a port called BOAT_PORT to pass records to DATATRIEVE, using the following steps:

1. Declares the port BOAT_PORT
2. Passes a STORE statement to DATATRIEVE, using the port
3. Prompts the user to input each field of the port
4. Calls DTEOF to end the STORE statement, when the user is finished entering records

Because data is entered from the terminal in ASCII format, all fields in the port are declared as character data types. Following is the DATATRIEVE command to declare the port BOAT_PORT:

```
DECLARE PORT BOAT_PORT USING
  01 BOAT.
    06 BUILDER PIC X(10).
    06 MODEL PIC X(10).
    06 RIG PIC X(6).
    06 LOA PIC XXX.
    06 DISP PIC XXXXX.
    06 BEAM PIC XX.
    06 PRICE PIC XXXXX.
```

The following Task Builder command file creates the task image for this program on RSTS/E systems:

```
ENTRY,ENTRY/-SP=ENTRY,
LB:C81LIB/LB,
LB:DTCLIB/LB:CICOB:NCRSTS:NOLC,
LB:DTCLIB/LB
/
UNITS=6
//
```

Following is the program ENTRY:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.            ENTRY.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.

01 YACHT.
     06 BUILDER PIC X(10).
     06 MODEL   PIC X(10).
     06 RIG     PIC X(6).
     06 LOA     PIC XXX.
     06 DISP    PIC 99999.
     06 BEAM    PIC 99.
     06 PRICE   PIC 99999.

01 BUILD-TXT    PIC X(9) VALUE "BUILDER: ".
01 MODEL-TXT    PIC X(7) VALUE "MODEL: ".
01 RIG-TXT      PIC X(5) VALUE "RIG: ".
01 LOA-TXT      PIC X(8) VALUE "LENGTH: ".
01 BEAM-TXT     PIC X(6) VALUE "BEAM: ".
01 DISP-TXT     PIC X(8) VALUE "WEIGHT: ".
01 PRICE-TXT    PIC X(7) VALUE "PRICE: ".
01 SPACE-TXT    PIC X VALUE " ".

************************************
*  Copy the DATATRIEVE Access Block  *
************************************

 COPY "DAB11.CBL".
```

```
01 NODE     PIC X(30).
01 MSGBUF   PIC X(80).
01 MSGLEN   PIC 9(4) COMP.
01 OPTIONS  PIC 9(4) COMP.
01 SHOWIT   PIC X(4).

01 CONT     PIC X.
01 COMMAND  PIC X(80).
01 DICT     PIC X(30).

PROCEDURE DIVISION.
010-INITIALIZE-INTERFACE.
        MOVE NOSEMI TO OPTIONS.
        DISPLAY "What node do you want to use?  " WITH NO ADVANCING.
        ACCEPT NODE.
        CALL "DTINIT" USING DAB STRLEN BUFLEN
                BY DESCRIPTOR NODE
                BY REFERENCE OPTIONS.
        DISPLAY "What dictionary would you like to use? "
            WITH NO ADVANCING.
        ACCEPT DICT.
        MOVE "SET DICTIONARY !CMD;" TO COMMAND.
        CALL "DTCMD" USING DAB
                        BY DESCRIPTOR COMMAND
                        DICT.
        PERFORM 900-PRINT-MESSAGES UNTIL
                DAB-W-STATE NOT = DTR-K-STATE-MSG AND
                DAB-W-STATE NOT = DTR-K-STATE-LINE.
        MOVE "READY YACHTS WRITE;" TO COMMAND.
        CALL "DTCMD" USING DAB
                        BY DESCRIPTOR COMMAND.
        PERFORM 900-PRINT-MESSAGES UNTIL
                DAB-W-STATE NOT = DTR-K-STATE-MSG AND
                DAB-W-STATE NOT = DTR-K-STATE-LINE.

        MOVE "DECLARE PORT BOAT_PORT USING 01 BOAT." TO COMMAND.
        CALL "DTCMD" USING DAB
                        BY DESCRIPTOR COMMAND.

        MOVE "    06 BUILDER PIC X(10)." TO COMMAND.
        CALL "DTCMD" USING DAB
                        BY DESCRIPTOR COMMAND.

        MOVE "    06 MODEL PIC X(10)." TO COMMAND.
        CALL "DTCMD" USING DAB
                        BY DESCRIPTOR COMMAND.

        MOVE "    06 RIG PIC X(6)." TO COMMAND.
        CALL "DTCMD" USING DAB
                        BY DESCRIPTOR COMMAND.

        MOVE "    06 LOA PIC XXX." TO COMMAND.
        CALL "DTCMD" USING DAB
                        BY DESCRIPTOR COMMAND.

        MOVE "    06 DISP PIC XXXXX." TO COMMAND.
        CALL "DTCMD" USING DAB
                        BY DESCRIPTOR COMMAND.

        MOVE "    06 BEAM PIC XX." TO COMMAND.
        CALL "DTCMD" USING DAB
                        BY DESCRIPTOR COMMAND.
```

```
        MOVE "    06 PRICE PIC XXXXX.;" TO COMMAND.
        CALL "DTCMD" USING DAB
                        BY DESCRIPTOR COMMAND.

        PERFORM 900-PRINT-MESSAGES UNTIL
                DAB-W-STATE NOT = DTR-K-STATE-MSG AND
                DAB-W-STATE NOT = DTR-K-STATE-LINE.

        MOVE SPACES TO COMMAND.
        MOVE "FOR BOAT_PORT STORE YACHTS USING BOAT = BOAT ;"
                TO COMMAND.
        CALL "DTCMD" USING DAB
                        BY DESCRIPTOR COMMAND.
        PERFORM 900-PRINT-MESSAGES UNTIL
                DAB-W-STATE NOT = DTR-K-STATE-MSG AND
                DAB-W-STATE NOT = DTR-K-STATE-LINE.
200-BEGINNING-OF-LOOP.
        PERFORM 600-GET-RECORD.
        CALL "DTPUTP" USING DAB BY DESCRIPTOR YACHT.
        PERFORM 900-PRINT-MESSAGES UNTIL
                DAB-W-STATE NOT = DTR-K-STATE-MSG AND
                DAB-W-STATE NOT = DTR-K-STATE-LINE.
        IF DAB-W-STATE = DTR-K-STATE-PUTP THEN GO TO 200-BEGINNING-OF-LOOP.
        DISPLAY "RECORD WAS NOT STORED.".
        DISPLAY "PRESS RETURN TO CONTINUE.".
        ACCEPT CONT.
        CALL "DTCMD" USING DAB
                        BY DESCRIPTOR COMMAND.
        GO TO 200-BEGINNING-OF-LOOP.

600-GET-RECORD.

        DISPLAY "Enter a carriage return after each field value.".
        DISPLAY "Enter ALL DONE to stop storing records.".

        DISPLAY BUILD-TXT WITH NO ADVANCING.
        ACCEPT BUILDER.
        IF BUILDER = "ALL DONE" GO TO 950-EOF.

        DISPLAY MODEL-TXT WITH NO ADVANCING.
        ACCEPT MODEL.
        IF MODEL = "ALL DONE" GO TO 950-EOF.

        DISPLAY RIG-TXT WITH NO ADVANCING.
        ACCEPT RIG.
        IF RIG = "ALL DONE" GO TO 950-EOF.

        DISPLAY LOA-TXT WITH NO ADVANCING.
        ACCEPT LOA.
        IF LOA = "ALL DONE" GO TO 950-EOF.

        DISPLAY BEAM-TXT WITH NO ADVANCING.
        ACCEPT BEAM.
        IF BEAM = "ALL DONE" GO TO 950-EOF.

        DISPLAY DISP-TXT WITH NO ADVANCING.
        ACCEPT DISP.
        IF DISP = "ALL DONE" GO TO 950-EOF.

        DISPLAY PRICE-TXT WITH NO ADVANCING.
        ACCEPT PRICE.
        IF PRICE = "ALL DONE" GO TO 950-EOF.
```

```
900-PRINT-MESSAGES.
        IF DAB-W-STATE = DTR-K-STATE-MSG
                CALL "DTMSG" USING DAB
                                BY DESCRIPTOR MSGBUF
                                BY REFERENCE MSGLEN
                DISPLAY MSGBUF
                IF DAB-W-ERR-SEV = SEV-K-SEVERE GO TO 999-EOJ.
        IF DAB-W-STATE = DTR-K-STATE-LINE
                CALL "DTLINE" USING DAB
                                BY DESCRIPTOR MSGBUF
                                BY REFERENCE MSGLEN
                DISPLAY MSGBUF.
        CALL "DTCONT" USING DAB.

950-EOF.
        CALL "DTPEOF" USING DAB.
        GO TO 999-EOJ.

999-EOJ.
        CALL "DTFINI" USING DAB.
        DISPLAY "END OF PROGRAM".
        STOP RUN.
```

6.2 A Sample Payroll Application

The program PAYROLL reads data from a file and creates two other files. The program calls DATATRIEVE to get employee information stored in a DATATRIEVE domain, HOURLY_LABOR.

The domain and record definitions for HOURLY_LABOR are as follows:

```
DOMAIN HOURLY_LABOR USING HOURLY_LABOR_REC ON LABOR.DAT;

RECORD HOURLY_LABOR_REC USING
01  PERSON.
    05 ID                    PIC IS 9(5).
    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 HOURLY_RATE              PIC IS 99.99
                               EDIT_STRING IS $$$.99.
```

The program uses a port to read in records from DATATRIEVE. The definition of the port is as follows:

```
PORT H_LABOR_PORT USING HOURLY_LABOR_REC;
```

The PAYROLL program does the following:

- Reads data from the file TIMECARD.DAT
- Uses H_LABOR_PORT to pass records from the HOURLY_LABOR domain to a record buffer
- Writes production data from TIMECARD.DAT to the file FINISHED.DAT
- Uses data from the DATATRIEVE record and TIMECARD.DAT to calculate the weekly employee salary
- Writes salary and other employee data to the file PAYROLL.LOG and displays that information at the terminal

The following Task Builder command file creates the task image on an RSX–11M–PLUS system:

```
PAYROLL,PAYROLL/-SP=PAYROLL,
LB:[1,1]C81LIB/LB,
LB:[1,1]DTCLIB/LB:CICOB:NC11M:NOLC,
LB:[1,1]DTCLIB/LB
/
UNITS=10
GBLPAT=PAYROLL:LUNMAP:177700:177777
//
```

Following is the program PAYROLL:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.            PAYROLL.

************************************************
* In this example, the program reads data      *
* from a sequential file and uses information  *
* from a DATATRIEVE domain to create log files.  *
************************************************

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.    PDP--11.
OBJECT-COMPUTER.    PDP--11.

INPUT-OUTPUT SECTION.

FILE-CONTROL.
    SELECT FINISHED-GOODS ASSIGN TO "FINISHED.DAT"
        FILE STATUS IS FNSH-GDS-STATUS.

    SELECT TIME-CARD-FILE ASSIGN TO "TIMECARD.DAT"
        FILE STATUS IS TIME-STATUS.

    SELECT PAYROLL-LOG-FILE ASSIGN TO "PAYROLL.LOG"
        FILE STATUS IS PAYROLL-STATUS.

DATA DIVISION.
FILE SECTION.
```

```
FD  FINISHED-GOODS.
01  FINISHED-REC.
    03  F-PRODUCT-NUMBER   PIC X(9).
    03  FILLER             PIC XX.
    03  F-JOB-HRS          PIC 999V9.
    03  FILLER             PIC XX.
    03  F-JOB-COST         PIC 9(4)V99.

FD  PAYROLL-LOG-FILE.
01  PAY-REC.
    03  P-EMPLOYEE-NUMBER  PIC 9(6).
    03  FILLER             PIC XXX.
    03  P-EMPLOYEE-NAME    PIC X(20).
    03  FILLER             PIC XXX.
    03  P-DEPT             PIC XXX.
    03  FILLER             PIC XXX.
    03  P-GROSS-PAY        PIC Z999V99.

FD  TIME-CARD-FILE
    RECORD VARYING FROM 18 TO 117 CHARACTERS
        DEPENDING ON RECORD-LENGTH.
01  TIME-REC.
    03  T-EMPLOYEE-NUMBER          PIC 9(5).
    03  T-JOB-COUNT                PIC 99.
    03  T-JOB-INFO  OCCURS 10 TIMES.
       05  T-PRODUCT-NUMBER        PIC X(9).
       05  T-PRODUCT-HRS           PIC 99.

WORKING-STORAGE SECTION.
01  NUMBER-STRING        PIC X(5) VALUE SPACES.
01  TIME-STATUS          PIC XX  VALUE SPACES.
01  FNSH-GDS-STATUS      PIC XX  VALUE SPACES.
01  PAYROLL-STATUS       PIC XX  VALUE SPACES.
01  MSG-LEN              PIC 9(9).
01  ERROR-CODE           PIC 9(9).
01  RECORD-LENGTH        PIC 999.
01  SUB1                 PIC 999 COMP VALUE ZEROES.
01  TOTAL-HOURS          PIC 99.
01  OVERTIME-PAY         PIC 9999V99.
01  GROSS-PAY            PIC 9999V99.
01  COUNTER              PIC 99.

01  LINENO               PIC 9.
01  COLNO                PIC 9.

01  PERSON.
    05 ID                PIC IS 9(5).
    05 EMPLOYEE-NAME.
        10 FIRST-NAME    PIC IS X(10).
        10 LAST-NAME     PIC IS X(10).
    05 DEPT              PIC IS XXX.
    05 HOURLY-RATE       PIC IS 99V99.
    05 SUP-ID            PIC IS 9(5).

************************************
* Copy the DATATRIEVE Access Block.  *
************************************

 COPY "DAB11.CBL".
```

```
*************************
* Declare the variables.  *
*************************

01 MSGBUF PIC X(80).
01 MSGLEN PIC 9(4) COMP.

01 NODE PIC X(31).
01 OPTIONS PIC 9(4) COMP.

01  COMMAND PIC X(80) VALUE "SET DICTIONARY !CMD;".
01  JOB-RATE PIC 99V99.
01  DICTNY PIC X(80).

PROCEDURE DIVISION.
000-OPEN-FILES.
        OPEN INPUT TIME-CARD-FILE.
        OPEN OUTPUT PAYROLL-LOG-FILE.
        OPEN OUTPUT FINISHED-GOODS.

010-INITIALIZE-INTERFACE.

*****************************************************************
*    Initialize the interface with DTINIT.  Use DTCMD           *
*    to ready domains and ports.                                *
*****************************************************************

        MOVE NOSEMI TO OPTIONS.
        DISPLAY "What node would you like to use? "
                WITH NO ADVANCING.
        ACCEPT NODE.
        CALL "DTINIT" USING DAB STRLEN BUFLEN BY DESCRIPTOR NODE
                        BY REFERENCE OPTIONS.
        DISPLAY "What dictionary would you like to use? "
                WITH NO ADVANCING.
        ACCEPT DICTNY.
        CALL "DTCMD" USING DAB
                        BY DESCRIPTOR COMMAND DICTNY.
        PERFORM 900-PRINT-MESSAGES UNTIL
                DAB-W-STATE NOT = DTR-K-STATE-MSG AND
                DAB-W-STATE NOT = DTR-K-STATE-LINE.

        MOVE "READY HOURLY_LABOR; " TO COMMAND.
        CALL "DTCMD" USING DAB
                        BY DESCRIPTOR COMMAND.
        PERFORM 900-PRINT-MESSAGES UNTIL
                DAB-W-STATE NOT = DTR-K-STATE-MSG AND
                DAB-W-STATE NOT = DTR-K-STATE-LINE.

        MOVE "READY H_LABOR_PORT WRITE;" TO COMMAND.
        CALL "DTCMD" USING DAB
                        BY DESCRIPTOR COMMAND.
        PERFORM 900-PRINT-MESSAGES UNTIL
                DAB-W-STATE NOT = DTR-K-STATE-MSG AND
                DAB-W-STATE NOT = DTR-K-STATE-LINE.
```

```
020-READ-TIME-CARD-FILE.
        READ TIME-CARD-FILE AT END
             GO TO 999-EOJ.
        GO TO 030-GET-EMPLOYEE-RECORD.
021-CONT.
        MOVE T-JOB-COUNT TO SUB1.
        MOVE ZEROES TO TOTAL-HOURS.
        PERFORM 040-STORE-FINISHED-GOODS UNTIL SUB1 = ZEROES.
        PERFORM 050-WRITE-PAYROLL-LOG.
        GO TO 020-READ-TIME-CARD-FILE.

030-GET-EMPLOYEE-RECORD.
*****************************************************************
*     Pass a DATATRIEVE command that will find all employees    *
*     with a given employee number.  Use a substitution         *
*     directive to pass the value in T-EMPLOYEE-NUMBER.         *
*****************************************************************

        MOVE T-EMPLOYEE-NUMBER TO NUMBER-STRING.
        MOVE "FOR HOURLY_LABOR WITH ID EQ !CMD" TO COMMAND.
        CALL "DTCMD" USING DAB
                        BY DESCRIPTOR COMMAND
                        NUMBER-STRING.
        PERFORM 900-PRINT-MESSAGES UNTIL
                DAB-W-STATE NOT = DTR-K-STATE-MSG AND
                DAB-W-STATE NOT = DTR-K-STATE-LINE.
        MOVE "STORE H_LABOR_PORT USING PERSON = PERSON;"
                        TO COMMAND.
        CALL "DTCMD" USING DAB
                        BY DESCRIPTOR COMMAND.
        PERFORM 900-PRINT-MESSAGES UNTIL
                DAB-W-STATE NOT = DTR-K-STATE-MSG AND
                DAB-W-STATE NOT = DTR-K-STATE-LINE.
        IF DAB-W-STATE NOT = DTR-K-STATE-GETP GO TO 100-NO-EMPLOYEE.
        CALL "DTGETP" USING DAB
                        BY DESCRIPTOR PERSON
                        BY REFERENCE RECORD-LENGTH.
        CALL "DTCONT" USING DAB.
        PERFORM 900-PRINT-MESSAGES UNTIL
                DAB-W-STATE NOT = DTR-K-STATE-MSG AND
                DAB-W-STATE NOT = DTR-K-STATE-LINE.
        GO TO 021-CONT.

040-STORE-FINISHED-GOODS.
*****************************************************************
*   Move the job-class, product-number, and the number of       *
*   hours worked into FINISHED-REC.  Write it out to the file.  *
*****************************************************************

        MOVE T-PRODUCT-NUMBER (SUB1) TO F-PRODUCT-NUMBER.
        MOVE T-PRODUCT-HRS (SUB1) TO F-JOB-HRS.
        MULTIPLY T-PRODUCT-HRS (SUB1) BY HOURLY-RATE GIVING
                F-JOB-COST.
        WRITE FINISHED-REC.
        ADD  T-PRODUCT-HRS (SUB1) TO TOTAL-HOURS.
        SUBTRACT 1 FROM SUB1.
```

```
050-WRITE-PAYROLL-LOG.
*****************************************************************
*  If hours are greater than 40 for hourly worker, add on the   *
*  overtime pay.  Move data into PAY-REC and write it           *
*  to the log file.                                             *
*****************************************************************

        MULTIPLY TOTAL-HOURS BY HOURLY-RATE GIVING GROSS-PAY.
        IF TOTAL-HOURS > 40 PERFORM 060-ADD-OVERTIME-PAY.
        MOVE T-EMPLOYEE-NUMBER TO P-EMPLOYEE-NUMBER.
        MOVE EMPLOYEE-NAME TO P-EMPLOYEE-NAME.
        MOVE DEPT TO P-DEPT.
        MOVE GROSS-PAY TO P-GROSS-PAY.
        WRITE PAY-REC.
        DISPLAY "Pay Record for Employee: ", P-EMPLOYEE-NUMBER.
        DISPLAY "    Name:       ", P-EMPLOYEE-NAME.
        DISPLAY "    Department: ", P-DEPT.
        DISPLAY "    Gross Pay:  ", P-GROSS-PAY.
        DISPLAY "    ".

060-ADD-OVERTIME-PAY.
        SUBTRACT 40 FROM TOTAL-HOURS.
        DIVIDE 2 INTO HOURLY-RATE.
        MULTIPLY TOTAL-HOURS BY HOURLY-RATE GIVING OVERTIME-PAY.
        ADD OVERTIME-PAY TO GROSS-PAY.

100-NO-EMPLOYEE.
*********************************************************
*  Alert operator if employee number is invalid.        *
*********************************************************
        PERFORM 900-PRINT-MESSAGES UNTIL
                DAB-W-STATE NOT = DTR-K-STATE-MSG AND
                DAB-W-STATE NOT = DTR-K-STATE-LINE.
        DISPLAY "NO EMPLOYEE WITH THIS NUMBER, CHECK IT".
        DISPLAY T-EMPLOYEE-NUMBER.
        GO TO 020-READ-TIME-CARD-FILE.

900-PRINT-MESSAGES.
        IF DAB-W-STATE = DTR-K-STATE-MSG
                CALL "DTMSG" USING DAB
                                   BY DESCRIPTOR MSGBUF
                                   BY REFERENCE MSGLEN
                DISPLAY MSGBUF
                IF DAB-W-ERR-SEV = SEV-K-SEVERE GO TO 999-EOJ.

        IF DAB-W-STATE = DTR-K-STATE-LINE
                CALL "DTLINE" USING DAB
                                   BY DESCRIPTOR MSGBUF
                                   BY REFERENCE MSGLEN
                DISPLAY MSGBUF.
        CALL "DTCONT" USING DAB.
```

```
999-EOJ.
*********************************************************
*    Shutdown interface, close files, and stop.         *
*********************************************************
        CALL "DTFINI" USING DAB.
        CLOSE FINISHED-GOODS.
        CLOSE TIME-CARD-FILE.
        CLOSE PAYROLL-LOG-FILE.
        DISPLAY "    ".
        DISPLAY "END OF PAYROLL UPDATE PROGRAM".
        STOP RUN.
```

Chapter 7

Sample BASIC Programs

This chapter contains several sample BASIC programs that call DATATRIEVE. These programs show how you can call DATATRIEVE to perform information management tasks.

7.1 Formatting a Report

The program COLUMNS creates a 2-column report of data in a DATATRIEVE domain. The program performs the following steps:

1. Prompts for the name of the domain, a record selection expression, and the names of the fields that you want in the report
2. Writes the record stream into a buffer (BIGBUF) in 2-column format
3. Displays the report on the screen

The program also contains an error handler to trap a CTRL/C entered at the terminal.

You may want to edit the program and change parameters, such as the number and width of columns and the number of lines per page. You can also modify the program to write the output to a file: include an OPEN statement and add PRINT #1% statements wherever PRINT statements occur in this program.

The following Task Builder command file creates the task image on an RSX–11M–PLUS system:

```
SY:COLUMNS/CP=SY:COLUMNS,
LB:[1,1]BP2OTS/LB,
LB:[1,1]DTCLIB/LB:CIBAS:NC11M:NOLC,
LB:[1,1]DTCLIB/LB
/
UNITS = 15
ASG = TI:13:15
ASG = SY:5:6:7:8:9:10:11:12
GBLPAT=COLUMNS:LUNMAP:001700:000000
EXTTSK= 512
//
```

Following is the program COLUMNS:

```
100     %INCLUDE 'DAB11.B2S'

        DECLARE WORD COUNTER, LENGTH, I, J, K, SEV

        COMMON (Buf) STRING     MSGBUF = 80%,      &
                                COMAND = 80%,      &
                                RSE = 80%,         &
                                LST = 80%,         &
                                NODE = 30%,        &
                                DICT = 30%,        &
                                DOMAIN = 30%,      &
                                BIGBUF (2%, 35%) = 35%, &
                                HEADERS (5%) = 35%


        ! Set up error handler to trap CTRL/C.
        ! The error handling is done at line 8000

        VARIABLE_X = CTRLC
        ON ERROR GOTO 8000

        PRINT "PRESS CTRL/C AT ANY TIME TO QUIT."
        PRINT

        ! Choose DECnet node.

500     LINPUT "What node would you like to use"; NODE
        CALL DTINIT (DAB, STRLEN, BUFLEN, NODE, NOSEMI)
        GOSUB Message

        IF DAB$W_STATE = DAB$K_STATE_INIT
        THEN
                 PRINT "Try another node or press CTRL/C to quit."
                 CALL DTFINI (DAB)
                 GOTO 500
        END IF

         ! Choose dictionary directory.

550     LINPUT "What dictionary would you like to use"; DICT
        COMAND = "SET DICTIONARY !CMD;"
        CALL DTCMD (DAB, COMAND, DICT)
        GOSUB Message

        IF SEV = SEV$K_ERROR
        THEN
                PRINT "Error in dictionary name.  Try again."
                GOTO 550
        END IF
```

```
        ! Choose domain.

600     CALL DTCMD (DAB BY REF, "SHOW DOMAINS;")

        GOSUB Message
        LINPUT "What domain would you like to use"; DOMAIN
        CALL DTCMD (DAB, "READY !CMD;", DOMAIN)
        GOSUB Message

        ! Check for error in user response.

        IF SEV = SEV$K_ERROR
        THEN
                PRINT "Domain not ready.  Try another domain."
                GOTO 600
        END IF

        ! Prompt for an RSE and field names.

700     CALL DTCMD (DAB, "SHOW FIELDS !CMD;", DOMAIN)
        GOSUB Message

        PRINT "Add a record selection expression to this FIND command."
        LINPUT "FIND :"; RSE
        LINPUT "Now enter a list of field names"; LST

        ! Instruct DATATRIEVE to print the chosen fields.
        ! Use the Message subroutine to print the record stream.

        CALL DTCMD (DAB, "FOR !CMD PRINT !CMD;", RSE, LST)
750     IF DAB$W_STATE = DAB$K_STATE_MSG
        THEN
                SEV = DAB$W_ERR_SEV
                CALL DTMSG (DAB, MSGBUF, LENGTH)
                PRINT MSGBUF
                GOTO 8000 IF SEV = SEV$K_SEVERE
                GOTO 700 IF SEV = SEV$K_ERROR
                CALL DTCONT (DAB)
                GOTO 750
        END IF

        ! Check for errors.
        IF DAB$W_STATE = DAB$K_STATE_CMD
        THEN
                PRINT "Try again."
                GOTO 700
        END IF

800     CALL DTCONT (DAB)          ! Skip first blank line.

        ! Set the counter of header lines to 0.

        COUNTER = 0%

        ! Move the header lines into the header buffer.

        WHILE LENGTH <> 0
                CALL DTLINE (DAB, MSGBUF, LENGTH)
                HEADERS(COUNTER) = MSGBUF
                CALL DTCONT (DAB)
        NEXT
        CALL DTCONT (DAB)
```

```
2000    FOR I = 1% TO 2%
2100            FOR J = 1% TO 35%

                        IF DAB$W_STATE = DTR$K_STATE_LINE
                        THEN
                                CALL DTLINE (DAB, MSGBUF, LENGTH)
                                BIGBUF(I,J) = MSGBUF
                                CALL DTCONT (DAB)
                        ELSE BIGBUF(I,J) = "  "
2200            NEXT J

2300    NEXT I

2600    FOR K = 1% TO COUNTER
                IF BIGBUF(2%,1%) = "  "
                THEN
                        PRINT HEADERS(K)
                ELSE
                        PRINT HEADERS(K) + "  " + HEADERS(K)
2800    NEXT K
2875    PRINT "  "

2900    FOR J = 1% TO 35%
                IF BIGBUF(1%,J) = "  "
                THEN
                        GOTO 2950
                ELSE
                        PRINT BIGBUF(1%,J) + "  " + BIGBUF(2%,J)
2950    NEXT J

3200    GOSUB Message IF DAB$W_STATE = DTR$K_STATE_MSG

3300    ! Use the predefined BASIC constant FF (Form Feed)
        ! to move to the next page of output.

        IF DAB$W_STATE = DTR$K_STATE_LINE
        THEN
                PRINT FF
                GOTO 2000
        END IF
        GOTO 8000

   Message:
        SEV = SEV$K_SUCCESS

        WHILE (DAB$W_STATE = DTR$K_STATE_MSG) OR &
              (DAB$W_STATE = DTR$K_STATE_LINE)

        SELECT DAB$W_STATE
                CASE DTR$K_STATE_MSG
                        SEV = DAB$W_ERR_SEV
                        CALL DTMSG (DAB, MSGBUF, LENGTH)
                        PRINT MSGBUF
                        GOTO 8000 IF DAB$W_ERR_SEV = SEV$K_SEVERE
                CASE DTR$K_STATE_LINE
                        CALL DTLINE (DAB, MSGBUF, LENGTH)
                        PRINT MSGBUF
        END SELECT

        CALL DTCONT(DAB)
        NEXT
        RETURN
```

```
7000    ! The error handler.

        IF ERR = 28
           THEN PRINT ">>> A CTRL/C was typed."
           ELSE PRINT ">>> An error has occurred."
        END IF

        PRINT ">>> Program ending."
        CALL DTUNWD (DAB)
        RESUME 8000

8000    CALL DTFINI (DAB)
8100    END
```

7.2 Calculating a Linear Regression Equation

The program LINEAR performs a linear regression on data from a DATATRIEVE domain. You can use this program to check whether two fields have a linear relationship, that is, whether there are numbers A and B such that FIELD1 = B * FIELD2 + A. The program performs the following steps:

1. Prompts the user for the names of a domain and two fields
2. Prompts for a DATATRIEVE FIND command
3. Determines which records are used in the regression, using the FIND command
4. Determines the regression coefficients and displays them at the terminal
5. Enables the user to see how close the relationship is to being linear, by displaying the actual and estimated field values

Note the use of COUNTERBUF in this program. The routine DTGETP must use an ASCII string parameter to retrieve values in a port. However, the counter in the program must be an integer. Therefore, the program retrieves a string, COUNTERBUF, from the port and maps it to a word integer, COUNTER, which the program uses.

The following Task Builder command file creates the task image on an RSX–11M–PLUS system:

```
SY:LINEAR/CP=SY:LINEAR,
LB:[1,1]BP2OTS/LB,
LB:[1,1]DTCLIB/LB:CIBAS:NC11M:NOLC,
LB:[1,1]DTCLIB/LB
/
UNITS = 15
ASG = TI:13:15
ASG = SY:5:6:7:8:9:10:11:12
GBLPAT=LINEAR:LUNMAP:001700:000000
EXTTSK= 512
//
```

Following is the program LINEAR:

```
100     ! DTR Definitions file goes here

        %INCLUDE "DAB11.B2S"

        DECLARE WORD RECLEN, LENGTH, SEV
        MAP (CT) STRING COUNTERBUF = 2%
        MAP (CT) WORD COUNTER
        MAP (AREA) REAL VALUE1, VALUE2
        MAP (AREA) STRING VALUES = 8%
        DECLARE SINGLE AVERAGES(2), &
                       SUMXY,       &
                       SUMX2,       &
                       TOP,         &
                       BOTTOM,      &
                       A,           &
                       B

        DECLARE INTEGER ANSWER

        COMMON (Buf) STRING  MSGBUF = 80%,  &
                             COMAND = 80%,  &
                             PORT = 80%,    &
                             NODE = 30%,    &
                             DICT = 30%,    &
                             DOMAIN = 30%,  &
                             FIELD1 = 30%,  &
                             FIELD2 = 30%

        LINPUT "What node would you like to use"; NODE
        CALL DTINIT (DAB, STRLEN, BUFLEN, NODE, NOSEMI)
        GOSUB Message

        COMAND = 'DECLARE PORT PT1 01 N PIC 9(5) COMP.;'
        CALL DTCMD (DAB, COMAND)
        GOSUB Message

        COMAND = "DECLARE PORT PT2 01 WHOLE."
        CALL DTCMD (DAB, COMAND)
        COMAND = "02 PART-A REAL. 02 PART-B REAL.;"
        CALL DTCMD (DAB, COMAND)
        GOSUB Message

        LINPUT "What dictionary would you like to use"; DICT
        COMAND = "SET DICTIONARY !CMD;"
        CALL DTCMD (DAB, COMAND, DICT)
        GOSUB Message
```

```
Ready:
     COMAND = "SHOW DOMAINS;"
     CALL DTCMD (DAB, COMAND)
     GOSUB Message
     LINPUT "What domain do you want to use"; DOMAIN
     COMAND = "READY !CMD;"
     CALL DTCMD (DAB, COMAND, DOMAIN)
     GOSUB Message

     IF SEV = SEV$K_ERROR
     THEN
             PRINT "READY failed.  Please try another domain."
             GOTO Ready
     END IF

Find_collection:
     PRINT "Please enter a command to form a collection"
     LINPUT COMAND
     CALL DTCMD (DAB, COMAND)
     GOSUB Message

     COMAND = "STORE PT1 USING N = COUNT;"
     CALL DTCMD (DAB, COMAND)
     GOSUB Message

     CALL DTGETP (DAB, COUNTERBUF, RECLEN)
     CALL DTCONT (DAB)
     GOSUB Message

     GOTO Find_collection IF COUNTER = 0%

!*****************************************************************&
!     FORMULAS USED TO FIND THE LINEAR EQUATION                   &
!                                                                 &
!     LINEAR EQUATION :  Y = B*X + A                              &
!                                                                 &
!     Equation to arrive at value for B:                          &
!                      (note:  E = summation                      &
!                              n = number of data elements used)  &
!                                                                 &
!             B = E(X*Y) - n(average(X) * average(Y))             &
!                 -----------------------------------             &
!                 E(X**2) - n(average(x)**2)                      &
!                                                                 &
!     Equation to arrive at value for A:                          &
!                                                                 &
!             A = average(Y) - (B * average(X))                   &
!                                                                 &
!*****************************************************************

Select_fields:
     COMAND = "SHOW FIELDS FOR !CMD;"
     CALL DTCMD (DAB, COMAND, DOMAIN)
     GOSUB Message

     LINPUT "What is the name of the independent field"; FIELD1
     LINPUT "What is the name of the dependent field"; FIELD2
```

```
        COMAND = "STORE PT2 USING BEGIN"
        CALL DTCMD (DAB, COMAND)
        COMAND = "PART-A = TOTAL !CMD"
        CALL DTCMD (DAB, COMAND, FIELD1)
        COMAND = "PART-B = TOTAL !CMD; END;"
        CALL DTCMD (DAB, COMAND, FIELD2)
        GOSUB Message
        CALL DTGETP (DAB, VALUES, RECLEN)
        CALL DTCONT (DAB)
        GOSUB Message

        SUMXY = 0.0
        SUMX2 = 0.0

        AVERAGES(1) = VALUE1 / COUNTER
        AVERAGES(2) = VALUE2 / COUNTER

        COMAND = "FOR CURRENT STORE PT2 USING BEGIN"
        CALL DTCMD (DAB, COMAND)
        COMAND = "PART-A = !CMD; PART-B = !CMD; END;"
        CALL DTCMD (DAB, COMAND, FIELD1, FIELD2)
        GOSUB Message

    Get_port:
        CALL DTGETP (DAB, VALUES, RECLEN)
        CALL DTCONT (DAB)
        SUMXY = SUMXY + (VALUE1 * VALUE2)
        SUMX2 = SUMX2 + (VALUE1 **2)
        GOTO Get_port IF DAB$W_STATE = DTR$K_STATE_GETP
        GOSUB Message

        TOP = (SUMXY - (COUNTER * AVERAGES(1) * AVERAGES(2)))
        BOTTOM = (SUMX2 - (COUNTER * AVERAGES(1)**2))

        B = TOP/BOTTOM
        A = AVERAGES(2) - ( B * AVERAGES(1) )

        PRINT "Best estimate for linear relation is..."
        PRINT FIELD2; " = "; A;" + ";B ;" * ";FIELD1

        INPUT "Enter 1 if you want to see relationship"; ANSWER
        GOSUB Show IF ANSWER = 1%

    Select_option:
        PRINT "Enter 1 to exit program"
        PRINT "Enter 2 to start over with new domain"
        PRINT "Enter 3 to start over with new collection"
        PRINT "Enter 4 to use same collection, different fields"

        INPUT D
        ON D GOTO Quit,              &
                  Ready,             &
                  Find_collection,   &
                  Select_fields      &
        OTHERWISE Invalid_entry

    Invalid_entry:
        PRINT "Invalid entry, try again"
        GOTO Select_option
```

```
Show:
     COMAND = 'FOR CURRENT PRINT !CMD, !CMD,'
     CALL DTCMD (DAB, COMAND, FIELD1, FIELD2)
     COMAND = '!CMD + !CMD * !CMD ("ESTIMATE");'
     CALL DTCMD (DAB, COMAND, STR$(A), STR$(B), FIELD1)

     GOSUB Message
     RETURN

     ! Message-handling subroutine:

Message:
     SEV = SEV$K_SUCCESS

     WHILE (DAB$W_STATE = DTR$K_STATE_MSG) OR &
           (DAB$W_STATE = DTR$K_STATE_LINE)

     SELECT DAB$W_STATE
             CASE DTR$K_STATE_MSG
                     SEV = DAB$W_ERR_SEV
                     CALL DTMSG (DAB, MSGBUF, LENGTH)
                     PRINT MSGBUF
                     GOTO Quit IF SEV = SEV$K_SEVERE
             CASE DTR$K_STATE_LINE
                     CALL DTLINE (DAB, MSGBUF, LENGTH)
                     PRINT MSGBUF
     END SELECT

     CALL DTCONT(DAB)
     NEXT
     RETURN
Quit:
     CALL DTFINI (DAB)
     END
```

Chapter 8

Reference Section

This chapter is a reference section describing each component of the DATATRIEVE–11 Call Interface. The previous chapters of this book tell you how these components work together and how to develop programs that use them. Use this section when you need specific information about a particular routine.

8.1 DATATRIEVE Access Block

Your program specifies a DATATRIEVE Access Block (DAB) to contain information that DATATRIEVE–11 must pass to the calling program. Table 8–1 shows the fields of the DAB.

Table 8–1: The DATATRIEVE Access Block

Field	Length	Description
DAB$W_IDI	1 word	Internal identifier. You do not need to access this value.
DAB$W_STATE	1 word	The state of the DATATRIEVE–11 interface. When DATATRIEVE returns from a routine call, this field contains a value specifying the new state. Table 8–2 provides more information on DATATRIEVE states.

(continued on next page)

Table 8–1 (Cont.): The DATATRIEVE Access Block

Field	Length	Description
DAB$W_ERR_CODE	1 word	A 2-byte value associated with a DATATRIEVE message.
DAB$W_ERR_SEV	1 word	A value (0 to 4) representing the severity of the error listed in DAB$W_ERR_CODE.
DAB$W_FLAGS	1 word	Information passed from the DATATRIEVE routine to the calling program.
DAB$W_STR_LEN	1 word	The length of a string passed by DATATRIEVE to the calling program. This is the length of the string in DAB$V_STRING.
DAB$V_RESERVE	20 bytes	Not used. This area is reserved for future use.
DAB$V_STRING	n1 bytes	A string returned by a DATATRIEVE routine. This field contains a prompt string, port name, or other string. The length of this buffer is passed as the second parameter in the DTINIT call. DAB$W_STR_LEN contains the actual length of the string stored in DAB$V_STRING. In the sample DAB inclusion files, the length of this field is 30 bytes.
DAB$V_BUFFER	n2 bytes	For internal use only. You should not access this field. The length of this field is passed as the third parameter to DTINIT. In the sample DAB inclusion files, the length of this field is 150 bytes.

The following sections explain some of these fields in more detail.

8.1.1 DATATRIEVE–11 States

This section and Section 8.2 describe the DATATRIEVE–11 states and routines. Section 8.2 describes the concept of the state and shows how states are used in your program. Note that states and routines are closely related. The current state determines the set of permissible or required routine calls, and the action of the current routine determines the state when the routine finishes executing. Therefore, the descriptions of the states include a list of the routines associated with them. Similarly, each routine description specifies the states associated with the routine. Table 8–2 describes the DATATRIEVE states.

Table 8–2: The DATATRIEVE States

DAB$W_state =	DATATRIEVE enters this state when:	DATATRIEVE expects one of the following actions:
DTR$K_STATE_INIT = 0	A DTINIT call fails.	Call DTINIT again to initialize the interface.
	DTFINI has executed successfully.	Call DTFINI.
DTR$K_STATE_CMD = 1	Waiting for the next command line.	Call DTCMD to pass DATATRIEVE a command line.
DTR$K_STATE_PVAL = 2	Waiting for the program to enter a value in response to a prompt.	Call DTPVAL to supply the value in response to the prompt.
DTR$K_STATE_LINE = 3	There is a print line ready for the program to display.	Call DTLINE to obtain the text of the print line. Call DTCONT to continue execution at the next state. This action is required.

(continued on next page)

Table 8–2 (Cont.): The DATATRIEVE States

DAB$W_state =	DATATRIEVE enters this state when:	DATATRIEVE expects one of the following actions:
DTR$K_STATE_MSG = 4	DATATRIEVE has a message which the program can retrieve. The message number and its severity have been placed in the DAB.	Call DTMSG to retrieve the message and place it in the program's buffer. Call DTCONT to continue execution to the next state. This action is required.
DTR$K_STATE_GETP = 5	DATATRIEVE has a record for the program to retrieve.	Call DTGETP to place a DATATRIEVE record in the record buffer. Call DTCONT to continue execution at the next state. This action is required.
DTR$K_STATE_PUTP = 6	Waiting for your program to pass a record to DATATRIEVE.	Call DTPUTP to pass a record to DATATRIEVE. Call DTPEOF to pass an end-of-file marker to DATATRIEVE.
All states		Call DTUNWD to unwind to DTR$K_STATE_CMD. Call DTFINI to end the DATATRIEVE session.

8.1.2 Error Codes and Error Severity

After entering the message state, DATATRIEVE places a 2-byte binary value in the DAB$W_ERR_CODE field of the DAB. You can test this value to detect specific errors. DATATRIEVE also places a severity code in the DAB$W_ERR_SEV field. Note that the even numbers signify error conditions and the odd numbers signify success conditions.

Table 8–3: The DATATRIEVE Error Severity Codes

DAB$W_ERR_SEV =	Severity is:
0	WARNING
1	SUCCESS
2	ERROR
3	INFORMATION
4	SEVERE ERROR

Your program can test for the severity of each error. A severity of ERROR usually means that the DATATRIEVE command or statement did not execute properly. Often, your program can recover from this kind of error by trying another DATATRIEVE command or statement. A severity of SEVERE ERROR sometimes means that an error occurred in the Call Interface. It also can occur if DATATRIEVE cannot continue because of an error in a subroutine call, such as the wrong number of arguments. It is a good idea to stop program execution in this case.

8.1.3 Flags

The bits in the field DAB$W_FLAGS contain information about the Call Interface. These flags allow your program to access that information. Table 8–4 shows the meaning of each flag bit.

Table 8–4: The Flags Field of the DATATRIEVE Access Block

Flag name	Value	This flag is set if:
DAB$M_DAB_ACTIVE	1	The DAB is initialized and the link to the DATATRIEVE server is established. Testing this flag serves much the same function as testing for a state of DTR$K_STATE_INIT.
DAB$M_DTR11	2	The interface is connected to a DATATRIEVE–11 server, not a VAX DATATRIEVE server.

(continued on next page)

Table 8–4 (Cont.): The Flags Field of the DATATRIEVE Access Block

Flag name	Value	This flag is set if:
DAB$M_PW_PROMPT	4	The state is DTR$K_STATE_PMPT and the prompt in the string buffer is a prompt for a password. You can test for this flag in order to suppress the echoing of the password when it is typed.
DAB$M_STR_OVERFLOW	8	The DAB string buffer, DAB$V_STRING, has overflowed.
DAB$M_BUF_OVERFLOW	16	The user's buffer has overflowed after a call to DTMSG, DTLINE, or DTGETP. That is, the message, print line, or record was too large to fit in the buffer the program declared for it.

8.1.4 The String Buffer

The DAB$V_STRING field (called the string buffer) of the DAB contains a prompt string or port name. The length of the string buffer is established by a parameter passed to DTINIT. When DATATRIEVE is waiting for a value or record to be passed, it places the appropriate prompt in this buffer. Your program can then display the contents of the buffer to prompt for the value. Table 8–5 shows these states and the contents of the string buffer for each.

Table 8–5: Contents of the DAB$V_STRING Field

State	Contents of DAB String Buffer
DTR$K_STATE_CMD	Command prompt string
DTR$K_STATE_PVAL	Value prompt string
DTR$K_STATE_GETP	Port name string
DTR$K_STATE_PUTP	Port name string

Some useful points in using the DAB$V_STRING field of the DAB are as follows:

- If a DATATRIEVE command causes a string to overflow the string buffer, the DAB$M_STR_OVERFLOW flag in DAB$W_FLAGS is set.

- The port name (for DTR$K_STATE_GETP and DTR$K_STATE_PUTP) is always truncated to 8 characters. If the length of the string buffer is less than 8 bytes, then the port name is truncated and the overflow flag is set.
- When DATATRIEVE places a string in the buffer, it places the string's length in the DAB$W_STR_LEN. If the string is shorter than the buffer, the buffer is filled on the right with blanks.

8.2 DATATRIEVE–11 Routines

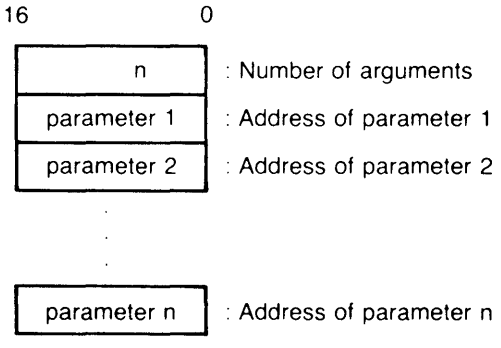
This section describes the callable DATATRIEVE routines. Each routine description includes a summary of the routine's function, the calling sequence (format), list of parameters, associated states, possible error messages, and a brief example. For an explanation of how to use the routine in a program, see Chapter 4. For complete examples in specific languages, see Chapters 5 through 7.

The Argument List

The argument list passed to a DATATRIEVE routine consists of a series of word addresses pointing to a set of parameters. The DATATRIEVE routines accept only two types of parameters: word integers and ASCII character strings. In the following routine descriptions, string parameters are enclosed in angle brackets. In languages that support string descriptors, such as COBOL and BASIC, the argument list entry for a string parameter is an address pointing to the descriptor for the string that you are passing to the routine. In languages that pass strings by specifying an address and a length, such as FORTRAN, you must include both values in the call to refer to the parameter. Thus, each parameter listed here in angle brackets is represented by two values in a FORTRAN program.

Figure 8–1 shows the structure of a FORTRAN argument list. Each line represents a 16-bit word.

Figure 8–1: Argument List for DATATRIEVE–11 Routines



If you are writing programs in a high-level language not discussed in this book or in MACRO–11 assembly language, design your interface to DATATRIEVE so that the argument list follows the FORTRAN conventions.

DTCMD

DTCMD passes a command line to DATATRIEVE. It is the main mechanism for executing a DATATRIEVE command or statement from your program. The command-str parameter can be a partial command or statement, a complete command or statement, or a series of commands and statements separated by semicolons.

After DTCMD executes, the string buffer in the DAB (DAB$V_STRING) contains the command prompt generated by the DATATRIEVE command or statement. For example, if DATATRIEVE executes a SET DICTIONARY command, it places the prompt remDTR> in the string buffer. If you have passed a partial command, DAB$V_STRING contains the CON> prompt.

Format

CALL DTCMD *(dab, <command-str> [, <arg-str> . . .])*

Parameters

dab
The DATATRIEVE Access Block for this call.

command-str
A DATATRIEVE command string. Passed by descriptor (COBOL and BASIC) or by an address and length (FORTRAN).

arg-str
A substitution string. When command-str contains the !CMD sequence, DATATRIEVE inserts the arg-str parameters in place of !CMD in the order that they appear in the parameter list. That is, the first arg-str is substituted for the first occurrence of !CMD and so on. This parameter is passed by descriptor (COBOL and BASIC) or by an address and length (FORTRAN). You cannot use more than five substitution strings in one call to DTCMD.

DTCMD

Associated States

- Call DTCMD when the state is DTR$K_STATE_CMD.
- The DATATRIEVE command that you pass to DTCMD determines the state after successful execution.

Examples

FORTRAN:

```
        CHARACTER*30  DOMAIN
        INTEGER*4     DOMLEN
              .
              .
              .
        WRITE (5,1000)
1000    FORMAT (' Enter the domain you want to use:  ',$)
        READ (5,2000) DOMLEN, DOMAIN
2000    FORMAT (Q,A)
100     CALL DTCMD (DAB, 'READY !CMD WRITE;', 17, DOMAIN, DOMLEN)
              .
              .
              .
```

COBOL:

```
Data division:

01  WS-COMMAND-LINE PIC X(80) VALUE "SET DICTIONARY !CMD;".
01  DICTNY PIC X(30).

Procedure division:

      DISPLAY "What dictionary would you like to use?  "
            WITH NO ADVANCING.
      ACCEPT DICTNY.
      CALL "DTCMD" USING DAB
                         BY DESCRIPTOR WS-COMMAND-LINE
                         BY DESCRIPTOR DICTNY.
```

BASIC:

```
130     COMMON (Buf) STRING   COMAND = 80%
            .
            .
            .
200     COMAND = "DECLARE PORT PT2 01 WHOLE."
        CALL DTCMD (DAB, COMAND)
        COMAND = "02 PART-A REAL. 02 PART-B REAL.;"
        CALL DTCMD (DAB, COMAND)
            .
            .
            .
```

DTCONT

If you call a DATATRIEVE routine that passes information to your program (DTLINE, DTMSG, DTGETP), the routine does not change the DATATRIEVE state. In these cases, you call DTCONT to continue. DTCONT simply causes DATATRIEVE to continue execution until it enters the next appropriate state.

For example, assume you pass a PRINT command to DATATRIEVE using DTCMD. Now you wish to retrieve the resulting print lines and display them. You must write a loop that includes the following:

1. A test for the state. Initially, the state is DTR$K_STATE_LINE.
2. A call to DTLINE. This call retrieves the next line from the print line buffer. After it executes, the state is still DTR$K_STATE_LINE.
3. A language statement to print the line, such as PRINT BUF.
4. A call to DTCONT. This call returns DATATRIEVE to the appropriate state.
   - If the state is still DTR$K_STATE_LINE, there are more lines to display, and looping continues.
   - If it is DTR$K_STATE_MSG, DATATRIEVE has placed a message in your program's buffer. You should check for success and exit from the loop.

Format

CALL DTCONT *(dab)*

Parameter

dab
The DATATRIEVE Access Block for this call.

Associated States

- You can call DTCONT when the state is one of the following:
  - DTR$K_STATE_MSG
  - DTR$K_STATE_LINE
  - DTR$K_STATE_GETP
- After DTCONT executes successfully, DATATRIEVE returns to a state determined by previous commands.

Example

This FORTRAN example shows a loop that calls DTLINE to retrieve a print line and place it in the buffer LINBUF. Then the program displays the line on the screen and calls DTCONT to proceed. This loop continues execution until DATATRIEVE reaches a different state. If there is no call to DTCONT, DATATRIEVE remains at the state DTR$K_STATE_LINE and puts the same line of text into LINBUF for each call to DTLINE.

```
20      IF (DABSTA .EQ. DBSLIN) THEN
                CALL DTLINE (DAB, LINBUF, 80, LEN)
                WRITE (5,*) LINBUF
                CALL DTCONT (DAB)
                GO TO 20
        END IF
```

DTFINI

Your program calls DTFINI to end the DATATRIEVE session. The routine works like the DATATRIEVE EXIT command. DTFINI finishes all domains, releases all collections, tables, and variables, and shuts down the DATATRIEVE Call Interface.

Format

CALL DTFINI *(dab)*

Parameter

dab
The DATATRIEVE Access Block for this call.

Associated States

- You can call DTFINI when DATATRIEVE is in any state.
- After DTFINI executes successfully, your program is no longer connected to DATATRIEVE.

Example

```
        .
        .
        .
1000    CALL DTFINI (DAB)
        END
```

DTGETP

You transfer records between your calling program and DATATRIEVE using ports. You define a port as a record buffer in your program. You also define the port in DATATRIEVE using the DEFINE PORT or DECLARE PORT command. Your program and DATATRIEVE can then access the port to send and receive records. Your program retrieves a record from a port using DTGETP. Note the following:

- If the record passed is shorter than the length of record-buf, DATATRIEVE does not use fill characters to fill the buffer.
- If the record is longer than the buffer, DATATRIEVE fills the buffer, truncates the record, and sets the flag DAB$M_BUF_OVERFLOW.

After DTGETP executes, the DAB$V_STRING buffer contains the name of the associated port.

Format

CALL DTGETP *(dab, <record-buf>, record-len)*

Parameters

dab
The DATATRIEVE Access Block for this call.

record-buf
The buffer to contain the port record. Passed by descriptor (COBOL and BASIC) or by an address and a length (FORTRAN).

record-len
DATATRIEVE places the length of the record passed to record-buf into this parameter.

DTGETP

Associated States

- You can call DTGETP when the state is DTR$K_STATE_GETP.
- After DTGETP executes successfully, the state is still DTR$K_STATE_GET Your program must call DTCONT to return to the next state.

Example

This COBOL example assumes that you have declared a port called EMP-PORT in DATATRIEVE and a corresponding record buffer called EMPLOYEE in your program. When the STORE statement has been executed successfully, DATATRIEVE enters the state DTR$K_STATE_GETP. Then the program calls DTGETP to retrieve the record and place it in the buffer. Finally, a call to DTCONT brings DATATRIEVE to the next state.

```
MOVE "FOR EMPLOYEES WITH EMP-NUM EQ !CMD" TO WS-COMMAND-LINE.
CALL "DTCMD" USING DAB
                   BY DESCRIPTOR WS-COMMAND-LINE
                   BY DESCRIPTOR T-EMPLOYEE-NUMBER.
MOVE "STORE EMP-PORT USING EMPLOYEE = EMPLOYEE;"
       TO WS-COMMAND-LINE.
CALL "DTCMD" USING DAB
                   BY DESCRIPTOR WS-COMMAND-LINE.
IF DAB-W-STATE NOT = DTR-K-STATE-PGET GO TO 100-NOCALL "DTGETP" USING DAB
                   BY DESCRIPTOR EMPLOYEE
                   BY REFERENCE RECORD-LENGTH.
CALL "DTCONT" USING DAB.
PERFORM 900-PRINT-MESSAGES UNTIL DAB-W-STATE
        NOT = DTR-K-STATE-MSG.
```

DTINIT

DTINIT initializes the DATATRIEVE Call Interface. It sets up the DATATRIEVE Access Block, establishes the DECnet node on which DATATRIEVE will run, and specifies a set of DATATRIEVE options.

There are two options available. If options is 1, then no semicolon is required at the end of a command or statement. If options is 2, then the standard DATATRIEVE banner is displayed on the terminal when DATATRIEVE is initialized at the remote or local node. A value of 0 disables both options, and a value of 3 enables both.

To activate the DATATRIEVE Remote Server on more than one DECnet node at once, declare a separate DAB and call DTINIT once for each node.

Format

CALL DTINIT *(dab, str-len, buff-len, <node-specification>, options)*

Parameters

dab
The DATATRIEVE Access Block for this call.

str-len
The length of the string buffer used internally by DATATRIEVE. This is the value in the DAB$W_STR_LEN field of the DAB. This value is set to 30 bytes in the DAB inclusion file. You may change it, but it should be set to at least 20 bytes.

buff-len
The length of the internal buffer DAB$V_BUFFER in bytes. This value is set to 150 bytes in the DAB inclusion file. You may change it, but it must be set to at least 132 bytes. The value you use for this parameter depends on the size of the records that your program handles:

- If you will be reading or writing records using DTGETP or DTPUTP, and those records are longer than 100 bytes, add 1 byte to buff-len for each byte in the record beyond 100.
- If you will be printing a line greater than 100 bytes, do the same as for records. Add 1 byte to buff-len for each byte beyond 100 in the longest print line.

That is, buff-len should be the largest of the following values:

- 132
- 32 + maximum record size
- 32 + maximum print line length

If buff-len is less than 132, DTINIT will generate an error message. If the buffer length is 132 or more, but the buffer is still not large enough to accommodate the record or print line, DATATRIEVE does not generate the error until it tries to place the record or print line in the buffer.

node-specification

The DECnet node specification that your program will use, if applicable (that is, if using the Remote Call Interface). This positional parameter must be blank if you intend to use the Local Call Interface. Only one Local Call Interface call is allowed in a program.

The node that you specify must have a DATATRIEVE server installed. The node specification is passed by descriptor (COBOL and BASIC) or by an address and a length (FORTRAN).

The syntax for a node specification is as follows:

```
node["account password"][::]
```

node	Is either the name of a DECnet node or blank.
account	Is the user name or account number.
password	Is the user's password.

options

A value representing a set of options that you can specify when initializing DATATRIEVE from your program. The following table lists the possible values of this parameter and the meaning of each.

Table 8–6: DTINIT Options

Option	Meaning
0	– Semicolons required. – DATATRIEVE banner is not displayed.
1	– Semicolons not required. – DATATRIEVE banner is not displayed.
2	– Semicolons required. – DATATRIEVE banner is displayed.
3	– Semicolons not required. – DATATRIEVE banner is displayed.

Associated States

- Call DTINIT before calling any other DATATRIEVE routine. Before you call DTINIT, DTR$W_STATE is 0 (unknown state).
- After DTINIT executes successfully, the state is normally DTR$K_STATE_CMD.

Examples

The variables NOSEMI and BANNER are defined as having the values 1 and 2, respectively, in the DAB definition file for each language. Appendix A lists the complete DAB definition files.

BASIC:

```
DECLARE WORD OPTIONS
COMMON (Buf) STRING     MSGBUF = 80%,     &
                        NODE = 30%,       &
                        DICT = 30%,       &
                        DOMAIN = 30%,     &

OPTIONS = NOSEMI + BANNER
INPUT "What node would you like to use"; NODE
CALL DTINIT (DAB, STRLEN, BUFLEN, NODE, OPTIONS)
```

COBOL variable declarations:

```
01 NODE     PIC X(6)  VALUE IS "BIGVAX".
01 OPTIONS  PIC 9(9)  USAGE IS COMP.
```

COBOL initialization call:

```
MOVE BANNER TO OPTIONS.
CALL "DTINIT" USING
        DAB STRLEN BUFLEN
        BY DESCRIPTOR NODE
        BY REFERENCE OPTIONS.
```

FORTRAN:

```
        CHARACTER*30 NODE
        INTEGER*4    NODLEN

        WRITE (5,1000)
1000    FORMAT (' Enter node specification:  ', $)
        READ (5,2000) NODLEN, NODE
2000    FORMAT (Q,A)
        CALL DTINIT (DAB, STRLEN, BUFLEN, NODE, NODLEN, NOSEMI)
```

DTLINE

When DATATRIEVE prints a line, your program can obtain the text of the line by calling DTLINE. DATATRIEVE places the print line text in the buffer you specify.

Format

CALL DTLINE *(dab, <pline-buf>, pline-len)*

Parameters

dab
The DATATRIEVE Access Block for this call.

pline-buf
A buffer to contain the print line text. The text is padded on the right with blanks. Passed by descriptor (COBOL and BASIC) or by an address and length (FORTRAN). If the line is too long for the buffer, DATATRIEVE sets the DAB$M_BUF_OVERFLOW bit in the flags field of the DAB.

pline-len
The length of the print line text, before padding.

Associated States

- You can call DTLINE when the state is DTR$K_STATE_LINE.
- After DTLINE executes successfully, DTR$K_STATE_LINE is still the state. Your program must call DTCONT to return to DTR$K_STATE_CMD.

Example

Section 4.5.2 contains an example for each language.

DTMSG

Your program must call DTMSG to obtain the text of a message generated by DATATRIEVE. When DATATRIEVE has an error message or an informational message to pass to your program, it places a binary error code and a severity code in the DAB$W_ERR_CODE and DAB$W_ERR_SEV fields of the DAB. Your program can check these fields for specific errors or for the severity of the current error and take appropriate action.

At this point, DATATRIEVE enters the state DTR$K_STATE_MSG. This state indicates that there is a message ready for the program to retrieve. These messages include error messages, informational messages (such as "Statement completed successfully."), and text resulting from SHOW commands.

Your program should call DTMSG to obtain the message text and place it in a buffer. After handling the message, your program can call DTCONT to continue.

NOTE

DAB$W_ERR_CODE and DAB$W_ERR_SEV are defined only if the current state is DTR$K_STATE_MSG. Therefore, if your program tests these fields of the DAB during any other DATATRIEVE state, the tests will be incorrect.

Format

CALL DTMSG *(dab, <msg-buff>, msg-len)*

Parameters

dab
The DATATRIEVE Access Block for this call.

msg-buf
A message buffer. When your program calls DTMSG, DATATRIEVE places the error message text in this buffer. This buffer is padded on the right with blanks. Passed by descriptor (COBOL and BASIC) or by an address and a length (FORTRAN).

msg-len
The true length of the error message text, before it is padded with blanks. If the message is too long for msg-buf, DATATRIEVE sets the DAB$M_BUF_OVERFLOW bit in the flags field of the DAB.

Associated States

- You call DTMSG when the state is DTR$K_STATE_MSG.
- After DTMSG executes, the state is still DTR$K_STATE_MSG. Your program must call DTCONT to return to the appropriate state.

Example

For a complete example of a message-handling subroutine in each language, see Section 4.5.2.

DTPEOF

When DATATRIEVE is receiving records by means of a declared port, your program calls DTPEOF to send an end-of-file marker to DATATRIEVE. When DTPEOF executes successfully, DATATRIEVE finishes executing the statement that is using the port and enters the state DTR$K_STATE_CMD.

Format

CALL DTPEOF *(dab)*

Parameter

dab
The DATATRIEVE Access Block for this call.

Associated States

- You can call DTPEOF when the state is DTR$K_STATE_PUTP to send an end-of-file marker to DATATRIEVE.
- After DTPEOF executes successfully, the state is DTR$K_STATE_CMD.

Example

This FORTRAN example assumes that the record buffer YACHT contains a complete record to pass to DATATRIEVE. If the user does not want to continue storing records and the state is DTR$K_STATE_PUTP, the program calls DTPEOF to pass the end-of-file marker to DATATRIEVE.

```
              .
              .
              .
      CALL DTPUTP (DAB, YACHT, 41)
      CALL MESAGE (SEV)

      WRITE (5,2200)
2200  FORMAT (' Do you wish to continue? [Y or N]  ', $)
      READ (5,3000) ANSWER
3000  FORMAT (A)
      IF ((ANSWER .EQ. 'Y') .OR. (ANSWER .EQ. 'y')) THEN
              GO TO 150
      END IF

200   IF (DABSTA .EQ. DBSPPU) CALL DTPEOF (DAB)
      CALL MESAGE (SEV)
      CALL DTFINI (DAB)

      WRITE (5,*) '           *****PROGRAM COMPLETED*****'
      END
```

DTPUTP

When you declare or define a port, you associate the name of the port with a record buffer declared in your program. Passing records to DATATRIEVE is then a 2-step process:

1. Call DTCMD, passing a DATATRIEVE statement that establishes a record stream using the port.
2. Call DTPUTP to pass a record from the program's record buffer through the port to DATATRIEVE. DATATRIEVE uses the port to associate a DATATRIEVE record structure with the contents of the record buffer.

Format

CALL DTPUTP *(dab, <record-buf>)*

Parameters

dab
The DATATRIEVE Access Block for this call.

record-buf
The record buffer in which your program stores the record to be passed to DATATRIEVE. Passed by descriptor (COBOL and BASIC) or by an address and a length (FORTRAN).

Associated States

- You can call DTPUTP when the state is DTR$K_STATE_PUTP.
- After DTPUTP executes successfully, DTR$K_STATE_PUTP is still the state. You must call DTPEOF to end the record stream or call DTPUTP to pass another record.

Example

In this COBOL program, the paragraph 600-GET-RECORD prompts the user for the fields of a record. These are placed in a record buffer called YACHT. The program then calls DTPUTP to pass YACHT to DATATRIEVE. Afterwards, it checks for messages and continues prompting for records. The complete program appears in Chapter 6.

```
200-BEGINNING-OF-LOOP.
        PERFORM 600-GET-RECORD.
        CALL "DTPUTP" USING DAB BY DESCRIPTOR YACHT.
        PERFORM 900-PRINT-MESSAGES UNTIL
                DAB-W-STATE NOT = DTR-K-STATE-MSG AND
                DAB-W-STATE NOT = DTR-K-STATE-LINE.
        IF DAB-W-STATE = DTR-K-STATE-PUTP THEN
                GO TO 200-BEGINNING-OF-LOOP.
        DISPLAY "RECORD WAS NOT STORED.".
        DISPLAY "PRESS RETURN TO CONTINUE.".
        ACCEPT CONT.
        CALL "DTCMD" USING DAB
                           BY DESCRIPTOR COMMAND.
        GO TO 200-BEGINNING-OF-LOOP.
```

DTPVAL

When DATATRIEVE executes a statement that contains a prompting expression, it enters DTR$K_STATE_PVAL. This state requires that you pass DATATRIEVE a value in response to the prompt. To do this, your program calls DTPVAL, using the value as the parameter. The value must be an ASCII string.

When DATATRIEVE is in the state DTR$K_STATE_PVAL, the prompt is placed in the DAB$V_STRING field of the DAB. If the prompt string is too large to fit in this field, the DAB$M_STR_OVERFLOW bit is set in the flags field.

Format

CALL DTPVAL *(dab, <value>)*

Parameters

dab
The DATATRIEVE Access Block for this call.

value
An ASCII string specifying the value to be passed in response to the DATATRIEVE prompt. Passed by descriptor (COBOL and BASIC) or by an address and length (FORTRAN).

Associated States

- Call DTPVAL when the state is DTR$K_STATE_PVAL.
- After DTPVAL executes successfully, it enters a state determined by previous calls to DATATRIEVE.

Example

The following BASIC code creates a collection and passes a MODIFY statement to DATATRIEVE. To provide a value for the field that the MODIFY statement specifies, the program displays the prompt string and calls DTPVAL.

```
      .
      .
      .
CALL DTCMD (DAB, "FIND YACHTS;")
CALL DTCMD (DAB, "MODIFY ALL RIG OF CURRENT;")

IF DTR$W_STATE = DTR$K_STATE_PVAL
  THEN
     PRINT DAB$V_STRING
     LINPUT FIELD_VALUE
     CALL DTPVAL (DAB, FIELD_VALUE)
END IF
```

DTUNWD

DTUNWD allows your program to abort commands. It discards the remainder of a command and returns DATATRIEVE to DTR$K_STATE_CMD. This routine allows you to stop executing a command at a prompt, as interactive DATATRIEVE does with CTRL/Z. It can also be used to allow the user to stop DATATRIEVE from printing records.

Format

CALL DTUNWD *(dab)*

Parameter

dab
The DATATRIEVE Access Block for this call.

Associated States

- You can call DTUNWD when DATATRIEVE is in any state.
- After DTUNWND executes successfully, the state is DTR$K_STATE_CMD.

Example

The following BASIC example illustrates how DTUNWD can be used to cancel a STORE command.

```
      .
      .
      .
CALL DTCMD (DAB, "STORE YACHTS;")
WHILE DTR$W_STATE = DTR$K_STATE_PVAL
   PRINT DAB$V_STRING
   PRINT "Enter a value or press RETURN to stop";FIELD_VALUE
   IF FIELD_VALUE = ""
      THEN  CALL DTUNWD (DAB)
      ELSE  CALL DTPVAL (DAB, FIELD_VALUE)
   END IF
NEXT
```

Appendix A

Definitions of the DATATRIEVE Access Block

This appendix contains the definitions of the DATATRIEVE Access Block in FORTRAN–77, COBOL–81, and BASIC-PLUS-2.

A.1 FORTRAN–77

```
C
C DATATRIEVE Access Block definitions -- FORTRAN-77
C
C
C DAB fields:
C
        INTEGER*2 DAB, DABSTA, DABERR, DABSEV, DABFLA, DABLEN
        LOGICAL*1 DABRES(20), DABSTR(30), DABBUF(150)
        COMMON /DAB/ DAB, DABSTA, DABERR, DABSEV,
        1 DABFLA, DABLEN, DABRES, DABSTR, DABBUF
C
C Assign values to the DTINIT parameters:
C
        INTEGER*4 STRLEN
        PARAMETER (STRLEN = 30)

        INTEGER*4 BUFLEN
        PARAMETER (BUFLEN = 150)
C
C Assign values to the DATATRIEVE states:
C
        INTEGER     DBSINI,
        1           DBSCMD,
        2           DBSPMT,
        3           DBSLIN,
        4           DBSMSG,
        5           DBSPGE,
        6           DBSPPU
```

```
        PARAMETER (DBSINI = 0,
        1          DBSCMD = 1,
        2          DBSPMT = 2,
        3          DBSLIN = 3,
        4          DBSMSG = 4,
        5          DBSPGE = 5,
        6          DBSPPU = 6)
C
C Assign values to the severity of errors:
C
        INTEGER    WARN,
        1          SUCCES,
        2          ERROR,
        3          INFOR,
        4          SEVERE

        PARAMETER (WARN   = 0,
        1          SUCCES = 1,
        2          ERROR  = 2,
        3          INFOR  = 3,
        4          SEVERE = 4)
```

A.2 COBOL–81

```
*************************************************
*                                               *
* DATATRIEVE Access Block Definitions -- COBOL-81 *
*                                               *
*************************************************

01  DAB.
     03 DAB-W-IDI      PIC 9(4) COMP.
     03 DAB-W-STATE    PIC 9(4) COMP.
     03 DAB-W-ERR-CODE PIC 9(4) COMP.
     03 DAB-W-ERR-SEV  PIC 9(4) COMP.
     03 DAB-W-FLAGS    PIC 9(4) COMP.
     03 DAB-W-STR-LEN  PIC 9(4) COMP.
     03 DAB-V-RESERVE  PIC X(20).
     03 DAB-V-STRING   PIC X(30).
     03 DAB-V-BUFFER   PIC X(150).

**********************************
* Parameters for the DTINIT call. *
**********************************

01 STRLEN PIC 9(4) COMP VALUE IS 30.
01 BUFLEN PIC 9(4) COMP VALUE IS 150.
01 NOSEMI PIC 9(4) COMP VALUE IS 1.
01 BANNER PIC 9(4) COMP VALUE IS 2.

***********
* States. *
***********
```

```
01 DTR-K-STATE-INIT  PIC 9(4) COMP VALUE IS 0.
01 DTR-K-STATE-CMD   PIC 9(4) COMP VALUE IS 1.
01 DTR-K-STATE-PVAL  PIC 9(4) COMP VALUE IS 2.
01 DTR-K-STATE-LINE  PIC 9(4) COMP VALUE IS 3.
01 DTR-K-STATE-MSG   PIC 9(4) COMP VALUE IS 4.
01 DTR-K-STATE-GETP  PIC 9(4) COMP VALUE IS 5.
01 DTR-K-STATE-PUTP  PIC 9(4) COMP VALUE IS 6.

****************************************
* Severity -- values for DAB-W-ERR-SEV  *
****************************************

01 SEV-K-WARNING    PIC 9(4) COMP VALUE IS 0.
01 SEV-K-SUCCESS    PIC 9(4) COMP VALUE IS 1.
01 SEV-K-ERROR      PIC 9(4) COMP VALUE IS 2.
01 SEV-K-INFO       PIC 9(4) COMP VALUE IS 3.
01 SEV-K-SEVERE     PIC 9(4) COMP VALUE IS 4.
```

A.3 BASIC-PLUS-2

NOTE

The BASIC DAB definition declares the DATATRIEVE routines as external subroutines. This allows the compiler to check the number and data type of your arguments. The exception is DTCMD. Because DTCMD allows a variable length argument list, it is not declared in the DAB file. This way, you do not have to include null arguments for the five substitution strings.

```
!
!  The DATATRIEVE Access Block
!
!
! String-length and buffer-length parameters for DTINIT:
!
DECLARE WORD CONSTANT RESERV = 20%
DECLARE WORD CONSTANT STRLEN = 30%
DECLARE WORD CONSTANT BUFLEN = 150%
MAP (Acsblk) WORD DAB,          &
                 DAB$W_STATE,    &
                 DAB$W_ERR_CODE, &
                 DAB$W_ERR_SEV,  &
                 DAB$W_FLAGS,    &
                 DAB$W_STR_LEN,  &
             STRING DAB$V_RESERV = RESERV, &
                 DAB$V_STRING = STRLEN, &
                 DAB$V_BUFFER = BUFLEN
!
! Options parameter for DTINIT:
!
DECLARE WORD CONSTANT NOSEMI = 1%
DECLARE WORD CONSTANT BANNER = 2%
!
!
! DATATRIEVE states:
!
DECLARE WORD CONSTANT DTR$K_STATE_INIT  = 0%
DECLARE WORD CONSTANT DTR$K_STATE_CMD   = 1%
DECLARE WORD CONSTANT DTR$K_STATE_PVAL  = 2%
DECLARE WORD CONSTANT DTR$K_STATE_LINE  = 3%
DECLARE WORD CONSTANT DTR$K_STATE_MSG   = 4%
DECLARE WORD CONSTANT DTR$K_STATE_GETP  = 5%
DECLARE WORD CONSTANT DTR$K_STATE_PUTP  = 6%
!
! Error severity field:
!
DECLARE WORD CONSTANT SEV$K_WARNING     = 0%
DECLARE WORD CONSTANT SEV$K_SUCCESS     = 1%
DECLARE WORD CONSTANT SEV$K_ERROR       = 2%
DECLARE WORD CONSTANT SEV$K_INFO        = 3%
DECLARE WORD CONSTANT SEV$K_SEVERE      = 4%
!
! DATATRIEVE routines:
!
EXTERNAL SUB DTCONT (WORD)
EXTERNAL SUB DTGETP (WORD, STRING, WORD)
EXTERNAL SUB DTINIT (WORD, WORD, WORD, STRING, WORD)
EXTERNAL SUB DTLINE (WORD, STRING, WORD)
EXTERNAL SUB DTMSG  (WORD, STRING, WORD)
EXTERNAL SUB DTPEOF (WORD)
EXTERNAL SUB DTPUTP (WORD, STRING)
EXTERNAL SUB DTPVAL (WORD, STRING)
EXTERNAL SUB DTUNWD (WORD)
```

Index

A

Aborting commands, DTUNWD, 4–31, 8–30
Access Block, DATATRIEVE
  See DATATRIEVE Access Block
Accounts
  default DECnet, 2–1
  specifying to the Remote Terminal Interface, 2–1
Allocating LUNs, 3–9
Argument list, 8–7f

B

BASIC sample programs, 7–1 to 7–9
BUFLEN, parameter to DTINIT, 4–10

C

Call Interface, 1–5, 1–5f
  closing, 4–31
  creating menus, 1–7
  initializing, 4–11, 4–12, 8–17
  overview, 4–1
  procedures used with, 1–7
  tables used with, 1–7
  writing programs that use, 4–1 to 4–31
Calls to DATATRIEVE
  DTCMD, 4–13, 8–9
  DTCONT, 4–25, 8–12
  DTFINI, 4–31, 8–14
  DTGETP, 4–24, 8–15
  DTINIT, 4–12, 8–17
  DTLINE, 4–15, 8–21
  DTMSG, 4–16, 8–22
  DTPEOF, 4–26, 8–24
  DTPUTP, 4–26, 8–26
Calls to DATATRIEVE (cont'd.)
  DTPVAL, 4–20, 8–28
  DTUNWD, 4–31, 8–30
Closing the Call Interface, 4–31
!CMD, substitution directive, 4–13
COBOL sample programs, 6–1 to 6–11
Commands, passing, using DTCMD, 4–13, 8–9
Compiling programs that call DATATRIEVE, 3–2
Components of DATATRIEVE–11, 1–1 to 1–2
Continuing, DTCONT, 4–25, 8–12
Copying domains, 2–3 to 2–4

D

DAB
  See DATATRIEVE Access Block
DAB$M_BUF_OVERFLOW flag, 8–21
DAB$M_STR_OVERFLOW flag, 8–6
DAB$V_BUFFER, 8–17
DAB$V_STRING, 8–6t, 8–9
DAB$W_ERR_CODE, 4–17, 8–4
DAB$W_ERR_SEV, 4–17, 8–4, 8–22
DAB$W_FLAGS, 8–5t
DAB$W_STR_LEN, 8–7, 8–17
DAB fields, 8–5t
  DAB$V_BUFFER, 8–17
  DAB$V_STRING, 8–6t, 8–9
  DAB$W_ERR_CODE, 4–17, 8–4, 8–22
  DAB$W_ERR_SEV, 4–17, 8–4, 8–22
  DAB$W_STR_LEN, 8–7, 8–17
DATATRIEVE–11, components of, 1–1 to 1–2
DATATRIEVE Access Block (DAB), 4–1, 4–8t, 8–1, 8–1t
  declaring, 4–8
DATATRIEVE states, 4–1, 4–7 to 4–8, 8–3t
  list of, 4–7

DDMF
See Distributed Server
DDMF.TSK, 1–2
DECLARE PORT statement, 4–23
DECnet
default accounts, 2–1
node specification, 2–1, 8–18
using the Remote Call Interface, 1–5
using the Remote Terminal Interface, 1–3
DEFINE PORT command, 4–23
Distributed Server, 1–2, 1–3, 1–5f
using with the Remote Call Interface, 1–5
Domains
copying, 2–3 to 2–4
DTCLIB.OLB, DATATRIEVE object module library, 3–9
DTCLIB.OLB library, 1–2, 1–5
DTCMD routine, 8–9
format, 4–13
substituting variables with, 4–13
DTCONT routine, 4–25, 8–12
DTFINI routine, 4–31, 8–14
DTGETP routine, 4–24, 8–15
DTINIT routine, 4–10, 4–12, 8–17
options, 8–18t
DTLINE routine, 4–15, 8–21
DTMSG routine, 4–16, 8–22
DTPEOF routine, 4–26, 8–24
DTPUTP routine, 4–26, 8–26
DTPVAL routine, 8–28
DTR.TSK, 1–2
DTUNWD routine, 4–31, 8–30

E

Ending the DATATRIEVE session, DTFINI, 4–31, 8–14
End-of-file marker, passing, DTPEOF, 4–26, 8–24
Error messages, retrieving, DTMSG, 4–16, 8–22
Error severity codes, DAB$W_ERR_SEV, 4–17, 8–4
Error status codes, DAB$W_ERR_CODE, 4–17, 8–4
Event Flag Numbers, 3–9
EXIT command, 2–2
EXTRACT command, 2–4

F

FORTRAN sample programs, 5–1 to 5–18

I

Inclusion files, DAB, 4–10
Initializing the Call Interface, 4–11, 4–12, 8–17
Interactive DATATRIEVE, 1–2

L

LCDDMF.TSK, 1–2
LEFT_RIGHT allocation, 2–3
Local Call Interface, 1–1, 8–18
Local Server, 1–2
Logical Unit Numbers
allocating, 3–9
LUNMAP, area for LUN specification, 3–10
LUNs
See Logical Unit Numbers

M

MACRO–11 assembly language, 8–8
MAJOR_MINOR allocation, 2–3
Menu interface, example, 5–1
Menus
creating with the Call Interface, 1–7
Messages, obtaining, DTMSG, 4–16, 8–22

N

Node specification, 2–1, 8–18

O

Object module libraries, 3–9
Obtaining messages, DTMSG, 4–16, 8–22
Obtaining print lines, DTLINE, 4–15, 8–21
Obtaining records from DATATRIEVE, DTGETP, 4–24, 8–15
Options, for DTINIT, 8–17, 8–18t
Overlays, 3–12

P

Passing command lines to DATATRIEVE, DTCMD, 4–13, 8–9
Passing end-of-file marker, DTPEOF, 4–26, 8–24
Passing records from DATATRIEVE, DTGETP, 8–15
Passing records to DATATRIEVE, DTPUTP, 4–26, 8–26
Passing values to DATATRIEVE, DTPVAL, 4–20, 8–28

Passwords
specifying to the Remote Terminal Interface, 2–1
Ports, 8–15
definition of, 4–22
Print lines, obtaining, DTLINE, 4–15, 8–21
Procedures
using with the Call Interface, 1–7
Prompting expressions, 4–20

R

Records
passing to DATATRIEVE, DTPUTP, 4–26, 8–26
retrieving from DATATRIEVE, DTGETP, 4–24, 8–15
transferring, 4–22 to 4–30
REMDTR, 1–2, 1–3
Remote Call Interface
See Call Interface
Remote Terminal Interface, 1–1, 1–2
advantages of, 1–4
copying domains, 2–3 to 2–4
example, 2–2, 2–4
exiting, 2–2
invoking, 2–1
specifying an account, 2–1
specifying a node, 2–1
testing DATATRIEVE, 2–2 to 2–3
using REMDTR, 1–3
Retrieving messages, DTMSG, 4–16, 8–22
Retrieving print lines, DTLINE, 4–15, 8–21
Retrieving records from DATATRIEVE, DTGETP, 4–24, 8–15
Routines, 8–7 to 8–30
DTCMD, 4–13, 8–9
DTCONT, 4–25, 8–12
DTFINI, 4–31, 8–14
DTGETP, 4–24, 8–15
DTINIT, 4–12, 8–17
DTLINE, 4–15, 8–21
DTMSG, 4–16, 8–22
DTPEOF, 4–26, 8–24
DTPUTP, 4–26, 8–26
DTPVAL, 4–20, 8–28
DTUNWD, 4–31, 8–30
Running programs that call DATATRIEVE, 3–1 to 3–12

S

Severity codes, DAB$W_ERR_SEV, 4–17, 8–4
Stallpoints, 4–1
Status code, DAB$W_ERR_CODE, 4–17, 8–4
Stopping command execution, DTUNWD, 4–31, 8–30
STORE statement
using, to store into a port, 4–24
STRLEN, parameter to DTINIT, 4–10
Substitution directive (!CMD), 4–13, 8–9

T

Tables
using with the Call Interface, 1–7
Task Builder
using, with Callable DATATRIEVE, 3–4 to 3–8
Task Builder command file
example of, on RSTS, 3–4
example of, on RSX–11M–PLUS, 3–5
Terminal Interface
See Remote Terminal Interface
Transferring data, 4–15
Transferring records, 4–22 to 4–30

U

UNITS Task Builder qualifier, 3–10
Unwinding, DTUNWD, 4–31, 8–30

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-DEC-DEMO (800-332-3366) using a 1200- or 2400-baud modem. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local DIGITAL subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	————	Local DIGITAL subsidiary or approved distributor
Internal[1]	————	SDC Order Processing - WMO/E15 *or* Software Distribution Center Digital Equipment Corporation Westminster, Massachusetts 01473

[1]For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

DATATRIEVE–11
Call Interface Manual
AA–U050C–TC

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

What I like best about this manual is _____

_____

What I like least about this manual is _____

_____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35
110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

Do Not Tear - Fold Here

Cut Along Dotted Line

Reader's Comments

DATATRIEVE–11
Call Interface Manual
AA–U050C–TC

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

What I like best about this manual is _____

_____

What I like least about this manual is _____

_____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

o Not Tear - Fold Here and Tape

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

)o Not Tear - Fold Here

Cut Along Dotted Line

digital

Printed in U.S.A.