

VAX DATATRIEVE

Guide to Programming and Customizing

Order Number: AA-P863E-TE

March 20, 1992

This manual explains how to use the VAX DATATRIEVE Call Interface. It also describes how to create user-defined keywords and user-defined functions and how to customize DATATRIEVE Help and message text.

OPERATING SYSTEM: VMS Version 5.4 or higher

SOFTWARE VERSION: VAX DATATRIEVE Version 6.0


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.

© Digital Equipment Corporation 1984, 1985, 1987, 1989.

The following are trademarks of Digital Equipment Corporation:

ACMS	DIBOL	ULTRIX
ALL-IN-1	MASSBUS	UNIBUS
DATATRIEVE	PDP	VAX
DEC	P/OS	VAX CDD
DEC/CMS	Professional	VAX FMS
DEC/MMS	Rainbow	VAXcluster
DECforms	RALLY	VAXstation
DECintact	Rdb/ELN	VIDA
DECmate	Rdb/VMS	VMS
DECnet	ReGIS	VT
DECdecision	RSTS	Work Processor
DECUS	RSX	
DECwindows	RT	
DECwrite	TDMS	

The following are third-party trademarks:

IBM is a registered trademark of International Business Machines, Inc.

PostScript is a registered trademark of Adobe Systems, Inc.

Contents

Preface	ix
1 Introduction to Programming and Customizing	
1.1 Writing Programs That Call DATATRIEVE	1-1
1.1.1 How Writing Calling Programs Extends DATATRIEVE	1-3
1.1.2 How DATATRIEVE Extends Programming Languages	1-4
1.2 Customizing DATATRIEVE	1-4
2 Writing Programs That Call VAX DATATRIEVE	
2.1 Overview of DATATRIEVE Programming Calls	2-1
2.1.1 Declare the DATATRIEVE Access Block (DAB)	2-2
2.1.2 Initialize the Call Interface	2-2
2.1.3 Issue DATATRIEVE Commands and Statements	2-3
2.1.4 Closing the DATATRIEVE Interface	2-4
2.1.5 Compiling and Linking DATATRIEVE Programs	2-4
2.2 DATATRIEVE Stallpoints—Transfer of Control	2-5
2.3 The DATATRIEVE Access Block	2-8
2.3.1 DAB\$L_CONDITION—Condition Codes	2-9
2.3.2 Message Buffers	2-10
2.3.2.1 Messages DATATRIEVE Stores in the Message Buffer	2-10
2.3.2.2 Messages DATATRIEVE Stores in the Auxiliary Message Buffer	2-11
2.3.3 DAB\$W_STATE—Stallpoint Information	2-12
2.3.4 DAB\$W_REC_LEN—Record Lengths	2-12
2.3.5 Input/Output Channels	2-12
2.3.6 DAB\$L_OPTIONS—Initial Options	2-15
2.3.7 User-Defined Keyword Information	2-15
2.4 Using the DATATRIEVE Call Interface	2-17

2.4.1	Command Processing	2-17
2.4.2	Reading and Storing Records	2-17
2.4.3	Defining Your Own Keywords	2-18
2.4.4	Getting Information About DATATRIEVE Objects	2-18
2.4.5	Writing Data to a Log File	2-18
2.4.6	Handling Errors	2-19
2.4.7	Debugging Your Program	2-20

3 DATATRIEVE Call Reference

DTR\$COMMAND	3-2
DTR\$CONTINUE	3-9
DTR\$CREATE_UDK	3-12
DTR\$DTR	3-17
DTR\$END_UDK	3-23
DTR\$FINISH	3-25
DTR\$FINISH_WINDOWS	3-27
DTR\$GET_PORT	3-29
DTR\$GET_STRING	3-35
DTR\$INFO	3-41
DTR\$INIT	3-58
DTR\$LOOKUP	3-65
DTR\$PORT_EOF	3-69
DTR\$PRINT_DAB	3-73
DTR\$PUT_OUTPUT	3-76
DTR\$PUT_PORT	3-79
DTR\$PUT_VALUE	3-82
DTR\$UNWIND	3-85
DTR\$WINDOW_MSG	3-88
DTR\$WINDOW_OUTPUT	3-90
DTR\$WINDOWS	3-92

4 Adding Functions to DATATRIEVE

4.1	Using DATATRIEVE Functions	4-1
4.2	DATATRIEVE Functions and External Procedures	4-2
4.3	How to Add Functions to DATATRIEVE	4-3
4.3.1	Write and Compile Your Procedures	4-4
4.3.2	Insert the Procedure Object Files into the Object Library	4-5
4.3.3	Add the Function Definitions to DTRFNDxx.MAR	4-5
4.3.3.1	The Format of a Function Definition	4-5
4.3.3.2	Sample Function Definitions	4-11
4.3.4	Assemble and Debug DTRFNDxx.MAR	4-14
4.3.5	Replace DTRFNDxx.MAR in the Object Library	4-16
4.3.6	Relink the DATATRIEVE Shareable Image	4-16

5 Customizing DATATRIEVE Help Text

5.1	How to Change Help Text	5-1
5.1.1	Copy the DATATRIEVE Help Library File	5-1
5.1.2	Extract the Help Modules You Want to Change	5-2
5.1.3	Edit the Extracted Help Text	5-2
5.1.3.1	The Structure of Help Text Files	5-2
5.1.4	Replace the Changed Text in DTRHELP.HLB	5-2
5.1.5	Replace the Old Help Library File with the New One	5-3
5.2	How to Create New Help Text	5-3

6 Customizing DATATRIEVE Messages

6.1	DATATRIEVE Messages	6-1
6.1.1	DATATRIEVE Messages and the Call Interface	6-2
6.1.2	Messages in Interactive DATATRIEVE	6-2
6.2	How to Change DATATRIEVE Messages	6-3
6.2.1	Copy the Message Source File	6-3
6.2.2	Edit the Message Source File	6-4
6.2.2.1	Structure of the Message Source File	6-4
6.2.2.2	Changing the Message Text	6-5
6.2.2.3	Using FAO Directives to Format Message Text	6-6
6.2.2.3.1	Substitution Directives	6-7
6.2.2.3.2	Formatting Directives	6-8
6.2.2.3.3	Parameter Interpretation Directives	6-9
6.2.3	Compile the Message Source File	6-10
6.2.4	Link the Object File	6-10

6.2.5	Replace the Old DTRMSG.S.EXE File with the New One	6-11
-------	--	------

7 Customizing ADT and Guide Mode

7.1	Customizing ADT	7-1
7.1.1	Copy the ADT Text File	7-2
7.1.2	Modify ADT Text	7-2
7.1.2.1	Ready the ADT_TEXT Domain	7-2
7.1.2.2	Modify ADT Text Strings	7-2
7.1.2.3	Add ADT Text Strings	7-3
7.1.2.4	Messages During an ADT Session	7-5
7.1.3	Replace the Old ADT Text File with the New One	7-5
7.2	Customizing Guide Mode	7-5
7.2.1	Copy the DTRGUIDE.DAT File	7-7
7.2.2	Modify the Distribution of Commands and Statements	7-7
7.2.3	Replace the Old DTRGUIDE.DAT with the New One	7-8

8 Customizing DATATRIEVE Text

8.1	DATATRIEVE Text	8-1
8.1.1	Syntax Prompt Text	8-1
8.1.2	SHOW Text	8-2
8.1.3	Date Text	8-3
8.1.4	Default Edit Strings	8-3
8.1.5	Statistical Text	8-3
8.1.6	Keywords	8-4
8.2	How to Change DATATRIEVE Text	8-4
8.2.1	Copy the DATATRIEVE Text File	8-4
8.2.2	Edit the Text File	8-5
8.2.3	Assemble the Text File	8-6
8.2.4	Replace the Text Object File in DTRLIBxx.OLB	8-6
8.2.5	Relink the DATATRIEVE Shareable Image	8-6

9 Translating DATATRIEVE

9.1	Planning Your Translation	9-2
9.2	How to Translate DATATRIEVE	9-3
9.2.1	Translating Keywords	9-3
9.2.2	Translating Help Text	9-4
9.2.3	Translating Messages	9-6
9.2.4	Translating ADT	9-7
9.2.5	Translating the Remaining Text Elements	9-8
9.2.6	Translating the Names of Functions	9-9
9.2.7	Relinking the DATATRIEVE Shareable Image	9-10

A Definitions of the DATATRIEVE Access Block

A.1	Location of DATATRIEVE Access Blocks	A-1
A.2	Location of Sample Programs	A-2
A.3	Defining the DATATRIEVE Access Block in Other VAX Languages	A-3
A.3.1	Defining the Access Block	A-3
A.3.2	Defining the Message Buffers	A-7
A.3.3	Defining DATATRIEVE Constants	A-7

B DATATRIEVE Message Information

C Argument Data Types

Index

Examples

2-1	Sample Error Checking Routine in BASIC	2-20
-----	--	------

Figures

1-1	VAX DATATRIEVE Architecture	1-2
A-1	Structure of the DATATRIEVE Access Block	A-4

Tables

2-1	Stallpoints and Calls	2-7
2-2	Useful DAB Fields	2-8
2-3	Content of Message Buffers	2-11
2-4	Stallpoint Values	2-12
2-5	DATATRIEVE Keywords	2-16
3-1	Types of Context for User-Defined Keywords	3-13
3-2	DTR\$DTR Control Options	3-17
3-3	DTR\$DTR Terminal Server Options	3-18
3-4	Token Types for the DTR\$GET_STRING Call	3-35
3-5	Info-Code Options	3-43
3-6	DTR\$WINDOWS Control Options	3-92
3-7	DTR\$WINDOWS DECwindows Terminal Server Options	3-93
4-1	Common VAX Data Types	4-10
4-2	Input Argument Types and Clauses	4-10
6-1	Sample DATATRIEVE Messages	6-3
6-2	FAO Directives Used in DATATRIEVE Messages	6-6
7-1	Guide Mode Levels	7-6
A-1	DAB Definition Files in DTR\$LIBRARY	A-1
A-2	Sample Programs	A-2
A-3	Files Containing Sample Programs that Call DATATRIEVE	A-2
A-4	Fields of the DATATRIEVE Access Block	A-5
A-5	DATATRIEVE Access Block Constants	A-7
C-1	Atomic Data Types	C-1
C-2	String Data Types	C-4
C-3	Miscellaneous Data Types	C-4

Preface

This manual explains how to call the VAX DATATRIEVE services (also referred to in this manual as DATATRIEVE) from within programs written in high level languages such as VAX FORTRAN, VAX BASIC, VAX COBOL, VAX PASCAL, VAX PL/I, and VAX C. It also describes how to create user-defined keywords and user-defined functions and how to customize DATATRIEVE help and message text.

Intended Audience

This manual addresses experienced users of at least one programming language. Some familiarity with DATATRIEVE is also required.

Operating System Information

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

For information on the compatibility of other software products with this version of VAX DATATRIEVE, refer to the System Support Addendum (SSA) that comes with the Software Product Description (SPD). You can use the SPD/SSA to verify which versions of your operating system are compatible with this version of VAX DATATRIEVE.

Related Documents

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

- *VAX DATATRIEVE User's Guide*
Describes how to use DATATRIEVE interactively.
- *VAX DATATRIEVE Reference Manual*
Contains reference information for DATATRIEVE.

- Language reference manuals for FORTRAN, COBOL, BASIC, PASCAL, PL/I and C.

References to Products

The VAX DATATRIEVE documentation to which this manual belongs refers to the following products by their abbreviated names:

- VAX CDD/Repository software is referred to as CDD/Repository.
- VAX ACMS software is referred to as ACMS.
- VAX DATATRIEVE software is referred to as DATATRIEVE.
- VAX Rdb/ELN software is referred to as Rdb/ELN.
- VAX Rdb/VMS software is referred to as Rdb/VMS.
- VAX TDMS software is referred to as TDMS.
- VAX FMS software is referred to as FMS.
- DECforms software is referred to as DECforms.
- VIDA software is referred to as VIDA.

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

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

1

Introduction to Programming and Customizing

This manual describes how to customize VAX DATATRIEVE to suit your information management needs and how to use DATATRIEVE services from your applications. It explains how you can perform the following actions:

- Use the DATATRIEVE data management services in a program
- Write programs that set up an interface to DATATRIEVE for end users
- Add keywords and functions to DATATRIEVE
- Modify all of the DATATRIEVE on-line text

1.1 Writing Programs That Call DATATRIEVE

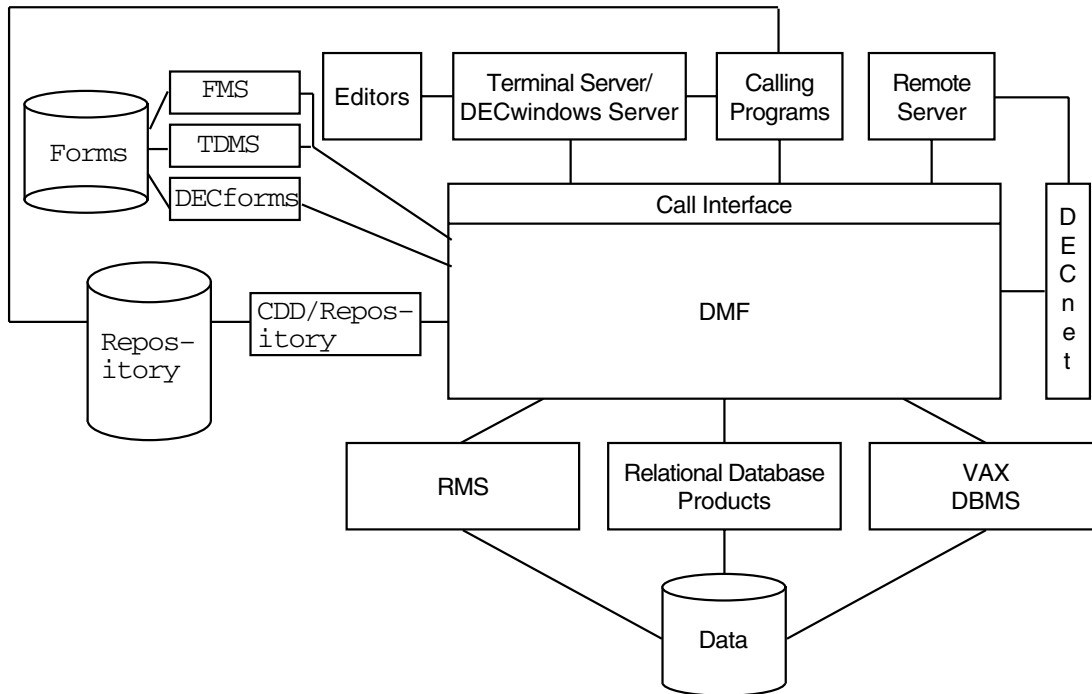
The DATATRIEVE Call Interface makes all of the DATATRIEVE information management capabilities available to calling programs written in languages such as VAX FORTRAN, VAX COBOL, VAX BASIC, VAX Pascal, VAX PL/I, and VAX C.

Figure 1–1 illustrates the role of calling programs and the Call Interface in the DATATRIEVE architecture.

Introduction to Programming and Customizing

1.1 Writing Programs That Call DATATRIEVE

Figure 1-1 VAX DATATRIEVE Architecture



The main part of the DATATRIEVE architecture is the Data Manipulation Facility (DMF). The DMF is the DATATRIEVE shareable image. It parses, compiles, and executes all commands and statements entered to DATATRIEVE. The DMF uses data definitions stored in the VAX CDD/Repository dictionary system to manipulate data managed by VAX RMS (Record Management Services), VAX DBMS, and DIGITAL relational database products such as VAX Rdb/VMS, VIDA, and VAX Rdb/ELN.

All requests for the DMF data management services go through the DATATRIEVE Call Interface. There are three modes of access to the DMF:

- DATATRIEVE terminal server

The most common way to use the DMF is with the DATATRIEVE terminal server. When you invoke DATATRIEVE with the DIGITAL Command Language (DCL) DATATRIEVE command, the terminal server gives you interactive access to the DMF data management services. The terminal server transmits your commands and statements through the Call Interface

Introduction to Programming and Customizing

1.1 Writing Programs That Call DATATRIEVE

to the DMF. The DMF returns print lines and messages, and the terminal server displays them.

- Remote server

A second mode of access to the DMF is the remote server. When you enter a command or statement that specifies a remote node, DATATRIEVE invokes the remote server on the node you specify. DATATRIEVE on the host node communicates with the DATATRIEVE remote server to process your commands and statements.

- User-written programs

The third mode of access to the DMF is a user-written program. Programs you write use the Call Interface to pass commands and statements to the DMF and to receive print lines and messages in return. When you write a program that calls the DMF, you create your own terminal server.

1.1.1 How Writing Calling Programs Extends DATATRIEVE

The Call Interface enables you to write programs that perform tasks that you cannot do by using the terminal server. Your programs can use the data retrieval services of DATATRIEVE and then perform complex calculations and statistical tests, set up complicated reports, and format terminal screens.

You can write programs that set up a menu-driven, end-user interface to information managed by DATATRIEVE, VAX DBMS, and relational database products. The Call Interface gives you access to the DATATRIEVE and Forms Interface, so your program can allow end users to display, store, and modify data using a full-screen form.

You can also write programs that define new keywords. For example, you may want to define a keyword, CORRELATE, that lets the user enter the following statement:

```
DTR> CORRELATE LOA, PRICE OF YACHTS WITH RIG = "SLOOP"
```

To create the user-defined keyword CORRELATE, you can write a program that calls the terminal server to simulate DATATRIEVE. The end user who runs this program cannot differentiate between your program and interactive DATATRIEVE. Your program can instruct DATATRIEVE to give it control when the user enters CORRELATE. The program can then parse the statement entered, perform the calculations required, and display the coefficient of correlation.

Introduction to Programming and Customizing

1.1 Writing Programs That Call DATATRIEVE

1.1.2 How DATATRIEVE Extends Programming Languages

VAX DATATRIEVE enables you to use data from a variety of sources. Records can be stored in a single RMS file, several RMS files (a DATATRIEVE view), a relational database, or a VAX DBMS database. You do not have to know how the records are stored to work with them. VAX DATATRIEVE performs all of the data access and your program does not have to specify how the records are structured.

You can pass entire DATATRIEVE commands and statements, including complicated record selection expressions, through the Call Interface. This allows you to specify the records you want to use when you run your programs, rather than when you write them. DATATRIEVE knows how the data file is organized and automatically searches for the records in the most efficient way.

The Call Interface gives your programs full access to DATATRIEVE functionality, including tables and procedures. If you use VAX DATATRIEVE when storing records, you can take advantage of the automatic validation and default value capabilities within DATATRIEVE.

1.2 Customizing DATATRIEVE

In addition to writing calling programs, you can customize DATATRIEVE in other ways.

You can extend the capability of DATATRIEVE to perform specific tasks efficiently by creating user-defined functions. For example, you may want to enable users to raise a number, 2, to a specified power, 9, with the following statement:

```
DTR> PRINT FN$POWER (2, 9)
```

FN\$POWER is a function you can define. You can also add functions that use Run-Time Library routines and system services.

Another way you can customize DATATRIEVE is by changing the DATATRIEVE on-line text. For example, you can translate all DATATRIEVE help text and messages into a foreign language.

If you customize DATATRIEVE, you may want to have multiple shareable images. For example, you may want a standard DATATRIEVE that you invoke with the command:

```
$ RUN DATATRIEVE
```

You may also want a German version invoked as follows:

```
$ RUN DATATRIEVE/VARIANT=GR
```

Introduction to Programming and Customizing 1.2 Customizing DATATRIEVE

To differentiate between multiple versions of DATATRIEVE, it is necessary to append a suffix such as GR to some of the DATATRIEVE file and image names. The following list identifies the files that you use to customize DATATRIEVE. The xx represents the optional characters at the end of each file name:

```
SYS$SHARE:DTRSHRxx.EXE
SYS$SYSTEM:DTR32xx.EXE
SYS$SYSTEM:DDMFxx.COM
SYS$SYSTEM:DDMFxx.EXE
DTR$LIBRARY:DTRBLDxx.COM
DTR$LIBRARY:DTRLIBxx.OLB
DTR$LIBRARY:DTRFNDxx.MAR
DTR$LIBRARY:DTRFUNxx.OLB
SYS$MANAGER:DTRSTUPxx.COM
```

Chapters 4, 5, 6, 7, 8, and 9 of this manual explain how to use a number of these files to customize DATATRIEVE. Refer to the *VAX DATATRIEVE Installation Guide* for more information about installing multiple copies of DATATRIEVE.

2

Writing Programs That Call VAX DATATRIEVE

This chapter explains how you can access the DATATRIEVE data management services from programs written in high-level languages such as FORTRAN, COBOL, BASIC, Pascal, PL/I, and C.

To write programs that call DATATRIEVE, you need to become familiar with:

- DATATRIEVE calls
- DATATRIEVE stallpoints
- The DATATRIEVE Access Block

The DATATRIEVE Access Block (DAB) is the storage section of your program.

This chapter discusses each of the three concepts separately. Chapter 3 provides reference material on each DATATRIEVE call in alphabetical order.

2.1 Overview of DATATRIEVE Programming Calls

Your program should perform five basic steps when calling DATATRIEVE routines:

1. Declare the DATATRIEVE Access Block (DAB).
2. Initialize the Call Interface by calling DTR\$INIT.
3. Issue DATATRIEVE commands and statements by calling DTR\$COMMAND, DTR\$DTR, or DTR\$WINDOWS.
4. Respond to the subsequent stallpoints and conditions by calling the routines that perform the functions you need.
5. Close the DATATRIEVE Interface by calling DTR\$FINISH or DTR\$FINISH_WINDOWS.

Writing Programs That Call VAX DATATRIEVE

2.1 Overview of DATATRIEVE Programming Calls

2.1.1 Declare the DATATRIEVE Access Block (DAB)

The DATATRIEVE Access Block (DAB) is a section of memory your program allocates for storing data and receiving information from DATATRIEVE. When you make a call to DATATRIEVE, DATATRIEVE uses the DAB to pass messages and information about the completion of the call to your program.

The DAB is a 100-byte data block that contains stallpoint information, condition codes, addresses of message buffers, and options that DATATRIEVE and your program use to communicate. In addition to this storage area, there is a DAB inclusion file containing the DAB definition and declarations of integer constants, message buffer sizes, and other information that your program can use.

The DATATRIEVE installation kit contains inclusion files that define the DAB in FORTRAN, BASIC, COBOL, Pascal, PL/I, and C. The files are located in the DTR\$LIBRARY directory. You can declare the DAB in your program by including the file DATATRIEVE provides ¹. For example, in a COBOL program you can include the following line:

```
COPY "DTR$LIBRARY:DAB.LIB".
```

Appendix A gives you more information on the names of the DAB inclusion files for each programming language.

Later sections in this chapter explain how you can use fields in the DAB to communicate with DATATRIEVE.

2.1.2 Initialize the Call Interface

The first DATATRIEVE call your program must make is to the DTR\$INIT routine. DTR\$INIT initializes various fields of the DAB and allocates internal data structures that DATATRIEVE uses. It also sets up the message buffer and the auxiliary message buffer and sets certain options, such as whether the Context Searcher is enabled. (Your program may change these options later.)

If initialization completes successfully, DATATRIEVE returns control to your program and waits for input at the command stallpoint (DTR\$K_STL_CMD). Your program should check the return status of the call to make sure that DTR\$INIT completed successfully before it continues.

¹ The inclusion file for C does not define the DAB, it only defines its type

Writing Programs That Call VAX DATATRIEVE

2.1 Overview of DATATRIEVE Programming Calls

2.1.3 Issue DATATRIEVE Commands and Statements

When DATATRIEVE reaches the command stallpoint (DTR\$K_STL_CMD), three calls are allowed: DTR\$COMMAND, DTR\$DTR, and DTR\$WINDOWS.

DTR\$DTR invokes the DATATRIEVE terminal server, allowing the user to interact with DATATRIEVE as though he or she were using interactive DATATRIEVE. Your program gains control when DATATRIEVE reaches certain stallpoints, which you specify. Otherwise, DATATRIEVE handles the stallpoints and conditions it encounters. For example, you can request that your program receive control only when the user enters a keyword defined by your program or terminates a statement with CTRL/Z or CTRL/C.

Note

A VAX BASIC program that calls DATATRIEVE cannot have the BASIC CTRL/C trapping enabled while DATATRIEVE is executing. Disable CTRL/C trapping in your program with the RCTLC function before the program calls DATATRIEVE. If the BASIC CTRLC function is enabled while DATATRIEVE is executing, pressing CTRL/C can produce unexpected results.

You can also use an optional argument to the DTR\$DTR call to tell DATATRIEVE to execute the commands in the startup file pointed to by the logical name DTR\$STARTUP. (See the description of the DAB\$M_OPT_STARTUP option to the DTR\$DTR call in Chapter 3.)

If you are using DATATRIEVE with DECwindows in a workstation environment, you should use the DTR\$WINDOWS call instead of the DTR\$DTR call. The DTR\$WINDOWS call performs the same function as the DTR\$DTR call, but in a DECwindows environment. With the DTR\$WINDOWS call, you can use all but three of the optional arguments that you use with DTR\$DTR. You cannot use the following:

- DTR\$M_OPT_BANNER
- DTR\$M_OPT_REMOVE_CTLC
- DTR\$M_OPT_KEYDEFS

See Chapter 3 for more information.

DTR\$COMMAND passes a command string to DATATRIEVE for processing. The string can be a partial command, a complete command or statement, or several commands or statements.

Writing Programs That Call VAX DATATRIEVE

2.1 Overview of DATATRIEVE Programming Calls

The commands and statements you pass with the DTR\$COMMAND call determine the resulting stallpoint. Your program should check the return status in DAB\$L_CONDITION and the stallpoint value in DAB\$W_STATE to decide what action to take next.

You should not use the DTR\$COMMAND call within a program loop to perform an iterative sequence of commands. Instead, your program will run more efficiently if you use a single call to DTR\$COMMAND using the DATATRIEVE REPEAT command to perform the looping.

2.1.4 Closing the DATATRIEVE Interface

When your program is finished using DATATRIEVE, it should call either the DTR\$FINISH call or the DTR\$FINISH_WINDOWS call to finish readied sources and release collections, tables, and variables; for example:

```
CALL DTR$FINISH (DAB)
```

Use the DTR\$FINISH_WINDOWS call instead of the DTR\$FINISH call if you are working with DATATRIEVE in a DECwindows environment.

2.1.5 Compiling and Linking DATATRIEVE Programs

Before you can run your program that calls DATATRIEVE, you must compile and link it.

You compile programs that call DATATRIEVE just as you would any normal program. For example, if you wrote the program in COBOL, you would invoke the COBOL compiler, specifying the file name:

```
$ COBOL DTRSAMPLE.COB
```

However, when you link the program, you must link it with the DATATRIEVE shareable image and, optionally, the DATATRIEVE terminal server.

To link your program, create an options file containing the following lines (in the examples in this book, the name of the options file is DTR.OPT):

```
SYS$SHARE:DTRSHR/SHARE, -  
DTR$LIBRARY:TERMSERVE/LIBRARY/INCLUDE=(ADT,EDT,GUI,HLP,LSE,TPU)
```

The first line links your program with the DATATRIEVE shareable image. The second line links your program with the DATATRIEVE terminal server library TERMSERVE.OLB and gives your program access to ADT, EDT, Guide Mode, help, VAX Language-Sensitive Editor (LSE), and VAXTPU.

Note

If the person installing DATATRIEVE at your site specifies a suffix for DATATRIEVE files during the installation, the file name for the

Writing Programs That Call VAX DATATRIEVE

2.1 Overview of DATATRIEVE Programming Calls

DATATRIEVE shareable image may be slightly different. Most often, the file name is of the form DTRSHRxx.EXE, with the optional suffix being a 2-character suffix. Note, however, that no such suffix is ever appended to the TERMSERVE.OLB file.

If your program does not call DTR\$DTR, DTR\$WINDOWS, or DTR\$PUT_OUTPUT, you can omit the second line of the options file. If you use ADT, Guide Mode, help, or any of the editors, you must include the appropriate module in the /INCLUDE list. If your program does not use ADT, EDT, Guide Mode, help, EDT, LSE, or VAXTPU, then you can eliminate the /INCLUDE clause.

After you create the options file, you can use the LINK command to link your program (and any separately compiled subroutines) to DTRSHR and TERMSERVE. For example, to run a COBOL program named ENTRY, use the following commands:

```
$ COBOL ENTRY
$ LINK ENTRY, DTR/OPT
$ RUN ENTRY
```

Note

You cannot use the /SERVICE_FAILURE qualifier with the RUN command when executing programs that call DATATRIEVE routines.

2.2 DATATRIEVE Stallpoints—Transfer of Control

Each DATATRIEVE call passes control from your program to DATATRIEVE. DATATRIEVE executes the commands and statements passed to it until it has a message or print line, or needs information to continue. When DATATRIEVE does not have a command to execute, it returns control to your program. Situations where DATATRIEVE waits for action by your program are called stallpoints.

Your program can identify the current stallpoint by examining the value of the DAB\$W_STATE field in the DAB. For example, after your program calls DTR\$INIT to initialize DATATRIEVE, the value of the DAB\$W_STATE field should be DTR\$K_STL_CMD, indicating that DATATRIEVE is at the command stallpoint.

There are nine DATATRIEVE stallpoints. The following list describes each stallpoint and the action necessary for DATATRIEVE to continue:

- The command stallpoint (DTR\$K_STL_CMD)

Writing Programs That Call VAX DATATRIEVE

2.2 DATATRIEVE Stallpoints—Transfer of Control

DATATRIEVE needs a command, or a command has been partially entered and DATATRIEVE needs the rest of the command. To continue, use the call DTR\$COMMAND, DTR\$DTR, or DTR\$WINDOWS.

- The prompt stallpoint (DTR\$K_STL_PRMP)

DATATRIEVE needs a value. This stallpoint occurs when DATATRIEVE is waiting for the user to respond to a prompt. To continue, use the call DTR\$PUT_VALUE, DTR\$DTR, or DTR\$WINDOWS.

- The print line stallpoint (DTR\$K_STL_LINE)

DATATRIEVE has a print line in the message buffer. This stallpoint occurs when DATATRIEVE executes a PRINT, LIST, or REPORT statement that does not specify a device or file to receive the print line. To continue, use the call DTR\$CONTINUE, DTR\$DTR, or DTR\$WINDOWS.

- The message stallpoint (DTR\$K_STL_MSG)

DATATRIEVE has a message in the message buffer. To continue, use the call DTR\$CONTINUE, DTR\$DTR, or DTR\$WINDOWS.

- The get port stallpoint (DTR\$K_STL_PGET)

DATATRIEVE has a record to pass to the host program. This stallpoint occurs when DATATRIEVE stores a record into a port. To continue, use the call DTR\$GET_PORT.

- The put port stallpoint (DTR\$K_STL_PPUT)

DATATRIEVE needs a record from the host program. This stallpoint occurs when DATATRIEVE evaluates a record selection expression (RSE) that specifies a port. To continue, use the call DTR\$PUT_PORT or DTR\$PORT_EOF.

- The continue stallpoint (DTR\$K_STL_CONT)

An interaction between DATATRIEVE and a program has occurred, and DATATRIEVE is ready to continue. To continue, use the call DTR\$CONTINUE, DTR\$DTR, or DTR\$WINDOWS.

This stallpoint occurs if your program set the DTR\$K_IMMED_RETURN option when it called DTR\$INIT. DATATRIEVE returns control to your program immediately after each call. See the description of DTR\$INIT in Chapter 3 for information on the Call Interface options.

- The user-defined keyword stallpoint (DTR\$K_STL_UDK)

DATATRIEVE needs the commands and statements that make up the keyword you are defining. This stallpoint occurs when a user enters a keyword defined with the call DTR\$CREATE_UDK. To continue, use the call DTR\$GET_STRING, DTR\$COMMAND, or DTR\$END_UDK.

Writing Programs That Call VAX DATATRIEVE

2.2 DATATRIEVE Stallpoints—Transfer of Control

- The end user-defined keyword stallpoint (DTR\$K_STL_END_UDK)
 A user-defined keyword declared as a statement has been entered. DATATRIEVE has ended processing of the keyword. To continue, use the call DTR\$END_UDK.

Table 2–1 summarizes the relationship of stallpoints and calls.

Table 2–1 Stallpoints and Calls

Stallpoint:	DATATRIEVE enters this stallpoint when:	To continue, use one of the following calls:
DTR\$K_STL_CMD	It is waiting for a command line.	DTR\$COMMAND DTR\$DTR DTR\$WINDOWS
DTR\$K_STL_PRMPPT	It is waiting for the user to enter a value in response to a prompt.	DTR\$PUT_VALUE DTR\$DTR DTR\$WINDOWS
DTR\$K_STL_LINE	It has a print line for the program to display.	DTR\$CONTINUE DTR\$DTR DTR\$WINDOWS
DTR\$K_STL_MSG	It has a message for the program to display.	DTR\$CONTINUE DTR\$DTR DTR\$WINDOWS
DTR\$K_STL_PGET	There is a record in the record buffer for the program to retrieve.	DTR\$GET_PORT
DTR\$K_STL_PPUT	It is waiting for the program to pass a record.	DTR\$PUT_PORT DTR\$PORT_EOF
DTR\$K_STL_CONT	The DTR\$K_IMMED_RETURN option is in effect. DATATRIEVE has responded to a call and is ready to continue.	DTR\$CONTINUE DTR\$DTR DTR\$WINDOWS
DTR\$K_STL_UDK	It is waiting for the commands and statements that make up a user-defined keyword.	DTR\$GET_STRING DTR\$COMMAND DTR\$END_UDK
DTR\$K_STL_END_UDK	It ends processing of a user-defined statement keyword.	DTR\$END_UDK

Writing Programs That Call VAX DATATRIEVE

2.3 The DATATRIEVE Access Block

2.3 The DATATRIEVE Access Block

The DATATRIEVE Access Block (DAB) is a section of memory your program allocates for storing data and receiving information from DATATRIEVE. When you first call DTR\$INIT, DATATRIEVE fills in many of the fields of the DAB and identifies the message buffers it will use to return messages. With each subsequent call, DATATRIEVE sets the values of fields in the DAB to indicate the current stallpoint, the completion status, and other information related to the call.

The first part of the DAB is a 100-byte data block that contains the stallpoint information, condition codes, addresses of message buffers, and flags that DATATRIEVE uses. In addition to this storage area, there is a DAB inclusion file containing the declarations of integer constants that your program can use to evaluate fields of the DAB.

Several fields in the DAB are reserved for internal use by Digital. However, other fields contain information that can be useful within your program. Table 2–2 lists the fields of the DAB that you can use and a brief explanation of the field's purpose.

Table 2–2 Useful DAB Fields

DAB Field	Contents of Field
DAB\$L_CONDITION	Numeric value identifying the status of the last DATATRIEVE command or statement
DAB\$A_MSG_BUF	Address of the message buffer
DAB\$W_MSG_BUF_LEN	Length of the message buffer
DAB\$W_MSG_LEN	Length of the current message
DAB\$A_AUX_BUF	Address of the auxiliary message buffer
DAB\$W_AUX_BUF_LEN	Length of the auxiliary message buffer
DAB\$W_AUX_LEN	Length of the current auxiliary message
DAB\$W_STATE	Value of the current stallpoint
DAB\$L_OPTIONS	Options value for the DTR\$INIT call
DAB\$W_REC_LEN	Length of the record DATATRIEVE passes to the program

(continued on next page)

Writing Programs That Call VAX DATATRIEVE

2.3 The DATATRIEVE Access Block

Table 2–2 (Cont.) Useful DAB Fields

DAB Field	Contents of Field
DAB\$W_TT_CHANNEL	Number of the input/output channel for ADT, Guide Mode, and help ¹
DAB\$L_COMMAND_KEYBOARD	Keyboard identifier for terminal server command input
DAB\$L_PROMPT_KEYBOARD	Keyboard identifier for terminal server input in response to prompts
DAB\$L_KEYTABLE_ID	Key table identifier for input key definitions
DAB\$W_UDK_INDEX	Number of the current user-defined keyword

¹Form managements (namely FMS, TDMS, DECforms) require a value different from zero to operate.

The following sections explain how your program can use these DAB fields.

2.3.1 DAB\$L_CONDITION—Condition Codes

Each time DATATRIEVE enters the message stallpoint (DTR\$K_STL_MSG), it stores a condition code in DAB\$L_CONDITION. This condition code indicates whether the last DATATRIEVE statement was completed, and what error occurred if the statement was not completed. In addition to storing the message code in DAB\$L_CONDITION, DATATRIEVE stores the associated message text in the DAB message buffer.

You can compare DAB\$L_CONDITION to condition names if you declare the condition names as external constants. For example, you can include the following statement in the working storage section of a COBOL program:

```
01 DTR$ _ERROR PIC 9(9) COMP VALUE IS EXTERNAL DTR$ _ERROR.
```

If you include this line, you can compare DTR\$ _ERROR to DAB\$L_CONDITION in the procedure division of your program. For example, you can use the following lines to check for DATATRIEVE commands and statements your program passes that are abandoned due to error:

```
IF      DAB$L_CONDITION = DTR$ _ERROR
THEN   DISPLAY "Error in DATATRIEVE command or statement."
```

A list of the DATATRIEVE condition codes and message texts is supplied on-line as part of the DATATRIEVE installation procedure. See Appendix B for information on how to access the on-line message documentation.

Writing Programs That Call VAX DATATRIEVE

2.3 The DATATRIEVE Access Block

2.3.2 Message Buffers

DATATRIEVE uses two buffers to pass print lines, messages, and other information to your program. These buffers are the message buffer and the auxiliary message buffer.

The DAB inclusion files declare the message buffer and the auxiliary message buffer as string variables. The sample DAB files use the names MSG_BUFF and AUX_BUFF to access the messages or lines DATATRIEVE passes to it. Your program can change the names and the lengths of both buffers by copying the sample DAB to your own directory and modifying it.

When you initialize the DATATRIEVE Call Interface, you can use the names of the buffers declared in the DAB, as in the following BASIC example:

```
CALL DTR$INIT (DAB BY REF, 100% BY REF, MSG_BUFF, AUX_BUFF, &  
              DTR$K_SEMI_COLON_OPT BY REF)
```

DATATRIEVE then stores the addresses of the buffers in the DAB\$A_MSG_BUF and DAB\$A_AUX_BUF fields of the DAB. It also stores the lengths of the buffers in the DAB\$W_MSG_BUF_LEN and DAB\$W_AUX_BUF_LEN fields. Each time DATATRIEVE uses one of the buffers, it looks in the DATATRIEVE Access Block to find the buffer's address.

You can display the content of the message buffer with a statement such as:

```
IF      DAB$L_CONDITION <> DTR$_SUCCESS  
THEN   PRINT MSG_BUFF  
END IF
```

This statement causes the program to display any message or line passed to it by DATATRIEVE other than the DTR\$_SUCCESS message.

2.3.2.1 Messages DATATRIEVE Stores in the Message Buffer

The information DATATRIEVE stores in the message buffer depends on the stallpoint. Table 2-3 shows the content of the message buffer corresponding to each stallpoint.

Writing Programs That Call VAX DATATRIEVE

2.3 The DATATRIEVE Access Block

Table 2-3 Content of Message Buffers

Stallpoint	Content of Message Buffer
DTR\$K_STL_CMD	Prompt to terminal, for example DTR>
DTR\$K_STL_PRMPPT	Prompt from DATATRIEVE prompt expression
DTR\$K_STL_LINE	A formatted print line
DTR\$K_STL_MSG	A formatted message
DTR\$K_STL_PGET	Name of port
DTR\$K_STL_PPUT	Name of port
DTR\$K_STL_CONT	Buffer is not used
DTR\$K_STL_UDK	Buffer is not used
DTR\$K_STL_END_UDK	Buffer is not used

DATATRIEVE stores the length of the message buffer in `DAB$W_MSG_BUF_LEN` and the length of the message in `DAB$W_MSG_LEN`. If the buffer is not long enough for the entire string, DATATRIEVE truncates the string. The buffer can also be zero characters long or omitted.

2.3.2.2 Messages DATATRIEVE Stores in the Auxiliary Message Buffer

For some types of messages, DATATRIEVE uses an auxiliary message buffer. Whenever a message contains an embedded string your program might use, DATATRIEVE stores the embedded string in the auxiliary buffer. For example, DATATRIEVE uses the auxiliary message buffer when executing the following DATATRIEVE statement:

```
ABORT "Collection is empty"
```

DATATRIEVE stores the complete message "ABORT: Collection is empty" in the message buffer and stores the string "Collection is empty" in the auxiliary message buffer. DATATRIEVE stores the length of the auxiliary message buffer in `DAB$W_AUX_BUF_LEN` and the length of the string in `DAB$W_AUX_LEN`. If the auxiliary message buffer is not long enough for the entire string, DATATRIEVE truncates the string. The buffer can also be zero characters long or omitted.

Writing Programs That Call VAX DATATRIEVE

2.3 The DATATRIEVE Access Block

2.3.3 DAB\$W_STATE—Stallpoint Information

To determine the current stallpoint, you must examine the value of the DAB\$W_STATE field. Table 2-4 shows the number corresponding to each stallpoint.

Table 2-4 Stallpoint Values

Value of DAB\$W_STATE	Stallpoint
1	DTR\$K_STL_CMD
2	DTR\$K_STL_PRMPPT
3	DTR\$K_STL_LINE
4	DTR\$K_STL_MSG
5	DTR\$K_STL_PGET
6	DTR\$K_STL_PPUT
7	DTR\$K_STL_CONT
8	DTR\$K_STL_UDK
9	DTR\$K_STL_END_UDK

The stallpoint names are declared as parameters in the inclusion file for FORTRAN, as constants in the inclusion file for C, BASIC, Pascal, and PL/I, and as 88-level condition names in the library file for COBOL.

2.3.4 DAB\$W_REC_LEN—Record Lengths

When you pass records from DATATRIEVE to your program, DATATRIEVE enters the get port stallpoint (DTR\$K_STL_PGET). Each time DATATRIEVE is at this stallpoint, it stores the length of the record it is passing in DAB\$W_REC_LEN. Your program can check the value of DAB\$W_REC_LEN to make sure that the record buffer your program uses in the DTR\$GET_PORT call is large enough.

2.3.5 Input/Output Channels

When you invoke DATATRIEVE, the terminal server translates the logical SYS\$INPUT to get the name of the device you are using. If you are using a terminal and DAB\$W_TT_CHANNEL has not been assigned yet, DATATRIEVE gets a channel number for the terminal and puts the number in DAB\$W_TT_CHANNEL.

The terminal server assigns a channel number whenever DAB\$W_TT_CHANNEL is zero and the calling program is being run from an interactive terminal. (The DAB\$W_TT_CHANNEL field is not used if the program is run in batch mode.) If you have already assigned a channel number in the DATATRIEVE Access Block, DATATRIEVE does not make a channel assignment. If your program assigns the

Writing Programs That Call VAX DATATRIEVE

2.3 The DATATRIEVE Access Block

channel number before calling the terminal server, then the number you assigned is not affected by later calls to the terminal server.

Note

DATATRIEVE does not assign a channel number to DAB\$W_TT_CHANNEL for FMS forms. FMS assigns a channel to SYS\$INPUT and uses that channel for FMS forms input and output.

The channel assigned in DAB\$W_TT_CHANNEL is the input/output channel for help, ADT, Guide Mode, and the TDMS Interface. Forms operations (using FMS, TDMS, and DECforms) take place only if DAB\$W_TT_CHANNEL contains a value different from zero. When a DECforms form is enabled, SYS\$INPUT is used as the display device without referring to the channel in DAB\$W_TT_CHANNEL,

If your program does not call the DATATRIEVE terminal server and you want to use forms, screen-oriented help, ADT, or Guide Mode, you must assign a channel number to DAB\$W_TT_CHANNEL. The simplest way to do this is to call the DATATRIEVE terminal server immediately after initializing DATATRIEVE:

```
CALL DTR$DTR (DAB, DTR$M_OPT_CMD)
```

The terminal server puts a channel number in DAB\$W_TT_CHANNEL and returns control to your program.

Each call to DTR\$DTR assigns channels for terminal input. Neither DTR\$FINISH nor DTR\$FINISH_WINDOWS deassigns these channels. You may find that after repeated calls to DTR\$INIT you may exceed the number of available channels. You can deassign channels after you call DTR\$FINISH or DTR\$FINISH_WINDOWS. Channel numbers are stored in the fields DAB\$W_TT_CHANNEL and DAB\$W_CTLC_CHANNEL in the DAB. The following lines represent a section of a FORTRAN program that shows how to deassign channels:

```
CALL DTR$FINISH(DAB)
C   Deassign the terminal channels
      status = sys$dassgn (%val(dab$w_tt_channel))
      status = sys$dassgn (%val(dab$w_ctlc_channel))
```

Note that the two lines that deassign the channel follow the call to DTR\$FINISH. You can use these two lines in any compatible program that calls DATATRIEVE.

Writing Programs That Call VAX DATATRIEVE

2.3 The DATATRIEVE Access Block

In addition to assigning a channel for ADT, forms, Guide Mode, and help, the DATATRIEVE terminal server uses the Run-Time Library Screen Management Facility (SMG) to perform command line input. (Note, however, that if you are using the DATATRIEVE DECwindows interface, the SMG interface is inactive and cannot be used with virtual keyboards.)

If your program is running interactively, DATATRIEVE creates two virtual keyboards: one for the command line and one for prompting expressions. The identifiers for these keyboards are stored in the fields DAB\$L_COMMAND_KEYBOARD and DAB\$L_PROMPT_KEYBOARD.

DATATRIEVE also creates a key definition table for both virtual keyboards and stores the key table identifier in the DAB\$L_KEYTABLE_ID field.

When entering DATATRIEVE commands, you can use CTRL/B and the arrow keys to recall previous commands. The default limit for the number of commands you can recall is 20. You can alter this limit to a number between zero and 100 with the logical name DTR\$COMMAND_LINES. To reset the recall limit for command input, define the logical name DTR\$COMMAND_LINES using either the DCL DEFINE or ASSIGN command before invoking DATATRIEVE. For example, the following command changes the recall limit to 30 lines:

```
$ DEFINE DTR$COMMAND_LINES "30"
```

Similarly, when responding to DATATRIEVE prompting expressions, you can recall previous responses. The default limit for the number of responses you can recall is 20. You can alter this limit to a number between zero and 100 by defining the logical name DTR\$PROMPT_LINES before invoking DATATRIEVE. For example:

```
$ DEFINE DTR$PROMPT_LINES "30"
```

Note that you must assign the command and prompt recall limits before invoking DATATRIEVE. Defining the logical DTR\$COMMAND_LINES or DTR\$PROMPT_LINES from within DATATRIEVE with FN\$CREATE_LOG has no effect. See the *VAX DATATRIEVE User's Guide* for more information on using the command recall function.

If you want your program to perform additional input using the same SMG key definitions as the DATATRIEVE terminal server, you can use the values in DAB\$L_COMMAND_KEYBOARD, DAB\$L_PROMPT_KEYBOARD, and DAB\$L_KEYTABLE_ID in calls to SMG routines.

SMG assigns two additional channels for DAB\$L_COMMAND_KEYBOARD and DAB\$L_PROMPT_KEYBOARD. You can deassign these channels by using the SMG call, SMG\$DELETE_VIRTUAL_KEYBOARD. Again, to deassign channels, you should do so after you have made either the DTR\$FINISH call or the DTR\$FINISH_WINDOWS call. The following lines represent a portion of code

Writing Programs That Call VAX DATATRIEVE

2.3 The DATATRIEVE Access Block

from a FORTRAN program that show you how to deassign these channels. Note that the two key lines deassigning the channels can be used in any program.

```
CALL DTR$FINISH(DAB)
C   Free SMG virtual keyboards
    status = smg$delete_virtual_keyboard (dab$1_command_keyboard)
    status = smg$delete_virtual_keyboard (dab$1_prompt_keyboard)
```

See the Run-Time Library documentation in the VMS documentation set for more information on the Screen Management Facility.

2.3.6 DAB\$L_OPTIONS—Initial Options

You can set various options when you initialize the DATATRIEVE Call Interface. For example, you can enable DATATRIEVE syntax prompting by specifying the DTR\$K_SYNTAX_PROMPT option in the DTR\$INIT call:

```
CALL DTR$INIT (DAB, 100, MSG_BUFF, AUX_BUFF, DTR$K_SYNTAX_PROMPT)
```

DATATRIEVE uses the DAB\$L_OPTIONS field of the DAB to store the options you specify. Your program can then use the DAB\$L_OPTIONS field to check what options are active.

Names for each of the initial options are defined in the DAB inclusion files. See the description of the DTR\$INIT call in Chapter 3 for a list of the initial options and their values.

2.3.7 User-Defined Keyword Information

A **user-defined keyword** (UDK) is a DATATRIEVE keyword you define and add to the DATATRIEVE language. You can write a program that uses the call DTR\$DTR or the call DTR\$WINDOWS to simulate interactive DATATRIEVE. In the program, you can use the DTR\$CREATE_UDK and DTR\$GET_STRING calls to define new keywords that perform tasks that interactive DATATRIEVE cannot do. The user who runs your program can use your UDKs in addition to the other DATATRIEVE commands and statements.

Each UDK is identified by a number. You specify this number when you create the UDK with the DTR\$CREATE_UDK call. When a user enters a UDK, DATATRIEVE returns its identifying number in the DAB\$W_UDK_INDEX field.

As described in the *VAX DATATRIEVE User's Guide*, you can use the file represented by the logical DTR\$SYNONYM to store synonyms for the DATATRIEVE keywords that you use frequently. However, you cannot use this file for creating synonyms for user-defined keywords.

Writing Programs That Call VAX DATATRIEVE

2.3 The DATATRIEVE Access Block

The DTR\$SYNONYM logical is evaluated at initialization time. Since user-defined keywords are defined by a callable program after initialization, DTR\$SYNONYM will not recognize any synonyms for the UDKs.

If you want to create synonyms for user-defined keywords, you can use the startup command file DTR\$STARTUP; however, keep in mind that the commands that you include in this file are executed the first time you call DTR\$DTR or DTR\$WINDOWS. This means that your callable program must make the necessary calls to DTR\$CREATE_UDK before calling DTR\$DTR or DTR\$WINDOWS.

You can find information on creating a startup command file in the *VAX DATATRIEVE User's Guide*.

Some DATATRIEVE keywords are defined internally in the same way as a user-defined keyword. These keywords are called DATATRIEVE keywords. When the user enters a DATATRIEVE keyword, DATATRIEVE passes its number in DAB\$W_UDK_INDEX. Table 2-5 shows the index number associated with each DATATRIEVE keyword.

Table 2-5 DATATRIEVE Keywords

Keyword	Index
EDIT	-4
ADT	-5
GUIDE	-6
EXIT	-7
@	-8
OPEN	-9
CLOSE	-10
HELP	-11
SHOW HELP	-12
SET HELP_PROMPT	-13
SET NO HELP_PROMPT	-14
SET HELP_WINDOW	-15
SET NO HELP_WINDOW	-16
SET HELP_LINES	-17

Writing Programs That Call VAX DATATRIEVE

2.4 Using the DATATRIEVE Call Interface

2.4 Using the DATATRIEVE Call Interface

The DATATRIEVE calls you use in your program and the order in which you use them depends on what you want to do. The following sections explain how to perform some common functions using the DATATRIEVE Call Interface. Refer to for extended examples of DATATRIEVE programming. See Appendix A and Chapter 3 for a complete description of individual DATATRIEVE calls.

2.4.1 Command Processing

DATATRIEVE provides the following calls to let your program process commands:

- DTR\$DTR or DTR\$WINDOWS returns control to DATATRIEVE until DATATRIEVE reaches the stallpoint you specify. By calling DTR\$DTR or DTR\$WINDOWS your program can invoke plots, ADT, and other interactive DATATRIEVE features.
- DTR\$COMMAND processes a command string.
- DTR\$PUT_VALUE returns a value in response to a prompt. When you pass DATATRIEVE a statement that stores or modifies fields or that contains a prompting expression, DATATRIEVE enters the prompt stallpoint (DTR\$K_STL_PRMP) and returns control to your program. DATATRIEVE cannot continue until you return a value with DTR\$PUT_VALUE.
- DTR\$CONTINUE resumes command execution.
- DTR\$UNWIND aborts the command being executed.

2.4.2 Reading and Storing Records

If your program wants to read records or store records using DATATRIEVE, it must first set up a record buffer to hold the records and a port to let DATATRIEVE and your program refer to the same record. You can set up the port in either of two ways:

- With the DEFINE PORT command. The name of the port and its associated record definition are stored in the dictionary.
- With the DECLARE PORT statement. The port created is temporary.

Your program can then:

- Call DTR\$GET_PORT at the DTR\$K_STL_PGET stallpoint to pass a record from DATATRIEVE to your program
- Call DTR\$PUT_PORT at the DTR\$K_STL_PPUT stallpoint to pass records from your program to DATATRIEVE

Writing Programs That Call VAX DATATRIEVE

2.4 Using the DATATRIEVE Call Interface

When you have no more records to pass to DATATRIEVE, your program calls DTR\$PORT_EOF at the DTR\$K_STL_PPUT stallpoint.

See the description of the DTR\$GET_PORT, DTR\$PORT_EOF, and DTR\$PUT_PORT calls in Chapter 3 for more information about using ports.

2.4.3 Defining Your Own Keywords

The DATATRIEVE calls DTR\$CREATE_UDK, DTR\$GET_STRING, and DTR\$END_UDK let your program add keywords to DATATRIEVE. In general, your program follows these steps to create and process user-defined keywords:

1. Uses DTR\$CREATE_UDK to create the keyword.
2. Calls DTR\$DTR or DTR\$WINDOWS with the DTR\$M_OPT_UDK option. DATATRIEVE will return control to your program whenever the user enters the keyword you have defined.
3. Uses DTR\$GET_STRING to process any arguments to the keyword.
4. Calls DTR\$END_UDK to tell DATATRIEVE you are done processing the keyword.

See the description of the DTR\$CREATE_UDK, DTR\$END_UDK, and DTR\$GET_STRING calls in Chapter 3 for more information about using user-defined keywords.

2.4.4 Getting Information About DATATRIEVE Objects

DATATRIEVE provides two calls that let you gather more information about DATATRIEVE objects:

- DTR\$LOOKUP returns a number that identifies the object about which you want information. This number is either the object-id if the object exists, or zero if the object is not found.
- DTR\$INFO returns detailed information, such as the object-id of a subschema in a VAX DBMS domain, a field's output picture string, or the number of arguments required by a particular plot.

See the description of the DTR\$INFO and DTR\$LOOKUP calls in Chapter 3 for more information explaining how to get information about objects.

2.4.5 Writing Data to a Log File

DTR\$PUT_OUTPUT writes a line to a file created by the DATATRIEVE command OPEN. See the description of the DTR\$PUT_OUTPUT call in Chapter 3 for more information.

Writing Programs That Call VAX DATATRIEVE

2.4 Using the DATATRIEVE Call Interface

2.4.6 Handling Errors

Every time you call a DATATRIEVE routine, it returns two values to indicate the status of the call. These values are as follows:

- The return status of the call itself
- The condition value returned in the DAB field DAB\$L_CONDITION

The return status that DATATRIEVE returns in register R0 indicates whether the call is valid and whether DATATRIEVE is able to interpret the arguments to the call. This value follows the standard format for VMS error condition codes; that is, you can evaluate success or failure by comparing the return status with the value SS\$_NORMAL:

```
RETURN_STATUS = DTR$COMMAND (DAB, "PRINT ALL YACHTS")
IF RETURN_STATUS <> SS$_NORMAL
THEN CALL LIB$SIGNAL (RETURN_STATUS BY VALUE)
END IF
```

Chapter 3 lists the errors that can result from each DATATRIEVE call.

However, the return status does not give you any indication whether DATATRIEVE succeeded at performing the requested action. For example, DATATRIEVE may succeed at interpreting the preceding call, but if you did not ready the YACHTS domain before issuing the call, DATATRIEVE cannot execute the command.

If DATATRIEVE encounters an error while performing a call, it enters the message stallpoint and stores the specific message text and value in the DAB\$A_MSG_BUF and DAB\$L_CONDITION fields of the DAB.

You should always examine the value of the DAB\$W_STATE field after each call to identify the current stallpoint. If the stallpoint is DTR\$K_STL_MSG, you should either evaluate the DAB\$L_CONDITION field to further identify the error or call DTR\$DTR or DTR\$WINDOWS with the DTR\$M_OPT_CMD option to allow DATATRIEVE to display the message text and handle the error condition.

In general, it is a good idea to check for both types of errors after every call, especially while you are debugging a program. Example 2-1 shows a BASIC subroutine that evaluates both the return status and the stallpoint to identify possible errors:

Writing Programs That Call VAX DATATRIEVE

2.4 Using the DATATRIEVE Call Interface

Example 2-1 Sample Error Checking Routine in BASIC

```
ERROR_CHECKING:
    ! Check for the validity of the call and its parameter.
    SELECT RETURN_STATUS
        CASE SS$NORMAL
            ! The call is valid; no action.
        CASE DTR$EXIT
            ! The user ended the session with CTRL/Z or EXIT.
            PRINT "DATATRIEVE session ended."
            GOTO DONE
        CASE ELSE
            ! All other errors are signaled.
            PRINT "A DATATRIEVE call has failed."
            CALL LIB$SIGNAL (RETURN_STATUS BY VALUE)
    END SELECT

    ! Check for the successful completion of the call.
    IF      (DAB$W_STATE = DTR$K_STL_MSG)           &
       AND (DAB$L_CONDITION <> DTR$_SUCCESS)
    THEN
        ! DATATRIEVE is at the message stallpoint
        ! and the message does not indicate success.
        ! Display the message on the screen and have
        ! DATATRIEVE return to the command stallpoint.

        PRINT MSG_BUFFER
        CALL DTR$CONTINUE (DAB BY REF)
    END IF
RETURN
```

2.4.7 Debugging Your Program

You can debug DATATRIEVE programs written in high-level languages just as you would any other program. The VMS Debugger allows you to display source code and examine variables as you step through the individual instructions of the program. (See the VMS documentation set for more information on the VMS Debugger.)

You can also debug DATATRIEVE programs by calling DTR\$PRINT_DAB to display the contents of the DATATRIEVE Access Block. For example, the following BASIC code calls DTR\$PRINT_DAB if the stallpoint from the DTR\$COMMAND call is not as expected:

Writing Programs That Call VAX DATATRIEVE 2.4 Using the DATATRIEVE Call Interface

```
LINPUT "What domain do you want to print"; DOMAIN
CALL DTR$COMMAND (DAB, "PRINT !CMD", DOMAIN)

IF    DAB$W_STATE = DTR$K_STL_LINE
THEN
    ! If DATATRIEVE is at the print line stallpoint,
    ! call DTR$DTR to have DATATRIEVE display the data.

    CALL DTR$DTR (DAB, DTR$M_OPT_CMD)
ELSE
    ! If not, display the contents of the DAB for analysis.

    PRINT "DATATRIEVE is not at the print line stallpoint."
    PRINT "The contents of the DAB are as follows:"
    CALL DTR$PRINT_DAB (DAB)
END IF
```

Sample Programs

Appendix A points you to directory locations where you can find sample FORTRAN, COBOL, BASIC, Pascal, PL/I, and C programs.

3

DATATRIEVE Call Reference

This chapter describes the DATATRIEVE calls and explains how you can use them to communicate with DATATRIEVE. For each call, the following information is provided:

- **Format**

The format specifies the placement of required and optional syntax elements for each call. The documentation conventions are listed at the front of this manual.

- **Arguments**

Arguments specify the data type, access method, passing mechanism, and content for each argument to the call.

- **Usage Notes**

Usage Notes explain the following:

- Requirements for using the call
- Results of the call
- Restrictions on using the call

- **Return Status**

The return status tells you the name and meaning of each status code DATATRIEVE returns to your program.

You can use each call as an external function in your program. This enables you to check the status the call returns to your program.

- **Examples**

The examples show representative uses of the call. Refer to Appendix A for more examples.

DTR\$COMMAND

DTR\$COMMAND

Passes a command string to DATATRIEVE.

Format

DTR\$COMMAND (dab, command-string [, p1, ...pn])

Arguments

dab

Data type: data block

Access: read/write

Mechanism: by reference

Is the DATATRIEVE Access Block.

Note

The C include file does not define the DAB, it only defines its type.

command-string

Data type: character string

Access: read-only

Mechanism: by descriptor

Is a DATATRIEVE command or statement. The command string can also be part of a command or statement or several commands or statements.

The command string can contain the following substitution directives:

!CMD

Inserts a string in the command string.

!VAL

Inserts a numeric value in the command string.

p1, ... pn

Data type: unspecified

Access: read-only

Mechanism: by descriptor

DTR\$COMMAND

Are the arguments for substitution directives. The number and order of arguments must be the same as the number and order of substitution directives in the command string. The data type of the argument depends on the substitution directive it replaces. An argument replacing the !CMD directive must be a character string data type. An argument replacing the !VAL directive can be any numeric data type or it can be a fixed-length character string data type containing a numeric string (for example, "25").

Usage Notes

- You can use this call when DATATRIEVE is at the command or user-defined keyword stallpoints (DTR\$K_STL_CMD or DTR\$K_STL_UDK).
- The commands and statements you pass with this call determine the resulting stallpoint. Most commands and statements cause the message stallpoint (DTR\$K_STL_MSG) to inform your program of the completion of the command or of an error. You can determine whether a message indicates an error or is only informational by checking the condition code stored in DAB\$L_CONDITION.
- Your program must handle the stallpoint that results from this call. For example, if you get the DTR\$_SUCCESS message, "Statement completed successfully," you should use the DTR\$CONTINUE, DTR\$DTR, or DTR\$WINDOWS call before passing your next DTR\$COMMAND call.
- For each !CMD and !VAL substitution directive in the command string, you must include an argument for DATATRIEVE to substitute.
- When storing a procedure in the dictionary, DATATRIEVE must ensure that each line of the procedure is no longer than 255 characters. The command string you pass to DATATRIEVE with the DTR\$COMMAND is restricted to 255 characters. However, each substitution directive within the command string can also be up to 255 characters in length.

To ensure that the procedure is stored with no more than 255 characters per line, DATATRIEVE isolates substitution strings as separate lines within the procedure. For example:

```
CALL DTR$COMMAND (DAB, "DEFINE PROCEDURE PRINT_YACHTS")
CALL DTR$COMMAND (DAB, "PRINT ALL YACHTS ON !CMD", "FOO.DAT")
CALL DTR$COMMAND (DAB, "END_PROCEDURE")
```

DTR\$COMMAND

The preceding calls result in a procedure that consists of two lines:

```
PRINT ALL YACHTS ON  
FOO.DAT
```

When you invoke the procedure, DATATRIEVE can execute it properly because the first line is not a complete command. However, consider the following code:

```
CALL DTR$COMMAND (DAB, "DEFINE PROCEDURE PRINT_YACHTS")  
CALL DTR$COMMAND (DAB, "PRINT ALL !CMD ON FOO.DAT", "YACHTS")  
CALL DTR$COMMAND (DAB, "END_PROCEDURE")
```

These calls result in a 3-line procedure:

```
PRINT ALL  
YACHTS  
ON FOO.DAT
```

DATATRIEVE cannot execute this procedure correctly because the first line — although only part of the intended command — is a complete and valid command. DATATRIEVE will interpret and execute the first line and then issue an error when it encounters YACHTS as a separate command line.

- The length of the command that you pass to DTR\$COMMAND can be up to 255 characters. If you need to pass a command string longer than 255 characters, you can set the DTR\$K_MORE_COMMANDS option in the DAB\$L_OPTIONS field of the DAB to delay DATATRIEVE command parsing and then pass parts of the command string in a series of calls to DTR\$COMMAND.

After you finish passing the command, reset the DAB\$L_OPTIONS field and call DTR\$COMMAND one last time to parse the completed command. Refer to Appendix A for a sample FORTRAN program (see the subroutine PARSE) that uses the DTR\$L_MORE_COMMANDS option to pass a long command string.

- If the command line you pass to DTR\$COMMAND invokes a command file, DATATRIEVE interprets the invocation operator "@" as a DATATRIEVE keyword and returns control to the program at the user-defined keyword stallpoint (DTR\$K_STL_UDK). To execute the command file, your program must either call DTR\$DTR or DTR\$WINDOWS to have DATATRIEVE handle the keyword or call DTR\$GET_STRING to get the file name, open the file, and pass the individual lines to DTR\$COMMAND.

DTR\$COMMAND

Return Status

SS\$_NORMAL

Call completed successfully.

DTR\$_BADHANDLE

The DAB is invalid.

DTR\$_BADNUMARG

Invalid number of arguments. At least two arguments (dab, command_string) are required.

DTR\$_BADSTRDES

Invalid string descriptor.

DTR\$_WRONGSTALL

Wrong stallpoint for this call. The stallpoint must be either DTR\$K_STL_CMD or DTR\$K_STL_UDK.

Other errors from RMS, system services, and Run-Time Library routines.

Examples

Pass a command to DATATRIEVE from a FORTRAN program:

```
STATUS = DTR$COMMAND (DAB, 'READY YACHTS;')
```

Pass a command to DATATRIEVE from a BASIC program:

```
STATUS = 800 DTR$COMMAND (DAB BY REF, "SHOW FIELDS FOR YACHTS;")
```

Pass a command to DATATRIEVE from a COBOL program:

```
MOVE SPACES TO COMMAND_LINE.  
MOVE "FINISH YACHTS;" TO COMMAND_LINE.  
C ALL "DTR$COMMAND" USING DAB  
  BY DESCRIPTOR COMMAND_LINE  
  GIVING STATUS.
```

Enter commands when you run your program:

```
10 WRITE (6, 10)  
   FORMAT (' Enter a command')  
   READ (5, 20) LINE  
20  FORMAT (A)  
   STATUS = DTR$COMMAND (DAB, LINE)
```

DTR\$COMMAND

Use substitution directives to change the values of arguments while you run a BASIC program:

```
2000 INPUT "Enter the name of the field: "; FIELD$
      INPUT "Enter the minimum value: "; VALUE%

      STATUS = DTR$COMMAND (DAB BY REF,"FIND YACHTS WITH &
      !CMD GE !VAL;", &
      FIELD$, &
      VALUE% BY DESC)
```

Use substitution directives to change the values of arguments while you run a Pascal program:

```
WRITE ('Enter the name of the field: ');
READLN (PARAM1);
WRITE ('Enter the minimum value: ');
READLN (PARAM2);
COMMAND := 'FIND YACHTS WITH !CMD GT !VAL;';
STATUS := DTR$COMMAND (DAB, COMMAND, PARAM1, PARAM2, PARAM3, PARAM4);
```

Write a FORTRAN program that enables a user to form and print out a record stream:

```
INCLUDE 'DTR$LIBRARY:DAB'
CHARACTER*80 LINE
CHARACTER*20 FIELD
CHARACTER*10 VALUE
INTEGER*4 DTR$INIT
INTEGER*4 DTR_OPTIONS
INTEGER RET_STATUS
LOGICAL NO_DICTIONARY/.TRUE./
EXTERNAL SS$NORMAL
```

C Declare options to be used in DTR\$DTR call.

```
DTR_OPTIONS = DTR$M_OPT_CMD + DTR$M_OPT_CONTROL_C
```

C Initialize the session with DATATRIEVE.

```
RET_STATUS = DTR$INIT (DAB, 100, MSG_BUFF, AUX_BUFF,
1 DTR$K_SEMI_COLON_OPT)
```

C Verify that the call was completed successfully.

```
IF (RET_STATUS .NE. %LOC(SS$NORMAL)) THEN
    WRITE (6, *) ' DATATRIEVE initialization failed.'
    STOP
END IF

DO WHILE (NO_DICTIONARY)
```

C Prompt the user for a SET DICTIONARY command.

DTR\$COMMAND

```
        WRITE (6, 25)
25      FORMAT (' Please enter a SET DICTIONARY command')
        READ (5, 50, END = 999) LINE
50      FORMAT (A)
C Pass the command to DATATRIEVE.
        RET_STATUS = DTR$COMMAND (DAB, LINE)
C Use the DTR$DTR call to handle messages and print lines from
C DATATRIEVE and return when at DTR$K_STL_CMD.
        RET_STATUS = DTR$DTR (DAB, DTR$M_OPT_CMD)
C Check to see if the SET DICTIONARY command was executed
C successfully. If it was not, start over.
        IF (DAB$L_CONDITION .EQ. %LOC(DTR$_SUCCESS)) THEN
            NO_DICTIONARY = .FALSE.
        ELSE
            WRITE (6, 75)
75      FORMAT (' Try again')
        END IF
    END DO
C Ready the YACHTS domain.
100     RET_STATUS = DTR$COMMAND (DAB, 'READY YACHTS')
C Print out any messages and return on command stallpoint.
        RET_STATUS = DTR$DTR (DAB, DTR$M_OPT_CMD)
        DO WHILE (.TRUE.)
C Show the fields available for the domain YACHTS.
            RET_STATUS = DTR$COMMAND (DAB, 'SHOW FIELDS YACHTS')
            RET_STATUS = DTR$DTR (DAB, DTR$M_OPT_CMD)
C Prompt for a field name and a value.
            WRITE (6, 125)
125     FORMAT (' Enter a field name for YACHTS (or CONTROL Z to
            1 exit): ', $)
            READ (5, 50, END = 999) FIELD
            WRITE (6, 150)
150     FORMAT (' Enter minimum value for the field: ', $)
            READ (5, 50, END = 999) VALUE
C Use substitution directives to form the record stream.
            RET_STATUS = DTR$COMMAND (DAB, 'PRINT YACHTS WITH !CMD GT !VAL',
            1 FIELD,
            2 VALUE)
C Use DTR$DTR to print out the record stream.
```

DTR\$COMMAND

```
        RET_STATUS = DTR$DTR (DAB, DTR_OPTIONS)
    END DO
999    RET_STATUS = DTR$FINISH (DAB)
    END
```

DTR\$CONTINUE

DTR\$CONTINUE

Instructs DATATRIEVE to continue processing.

Format

DTR\$CONTINUE (dab)

Argument

dab

Data type: data block

Access: read/write

Mechanism: by reference

Is the DATATRIEVE Access Block.

Usage Notes

- You can use the DTR\$CONTINUE call when DATATRIEVE is at the continue, message, or print line stallpoint (DTR\$K_STL_CONT, DTR\$K_STL_MSG, or DTR\$K_STL_LINE).
- When DATATRIEVE has a message, it stores the message text in the message buffer and the associated number in DAB\$L_CONDITION. For some types of messages, DATATRIEVE also stores a string in the auxiliary message buffer. DATATRIEVE then enters the message stallpoint (DTR\$K_STL_MSG). To restart from the message stallpoint, your program can use the call DTR\$CONTINUE.
- When DATATRIEVE executes a PRINT statement that does not specify a file or device to receive the output, DATATRIEVE stores a print line in the message buffer and enters the print line stallpoint (DTR\$K_STL_LINE). Your program receives control and can get the print line out of the message buffer. To clear the message buffer and get the remaining print lines, your program can use the call DTR\$CONTINUE.
- If you specify an output device or file in a PRINT statement, DATATRIEVE does not have to stall after each print line.
- If you set the DTR\$K_IMMED_RETURN option, DATATRIEVE stalls at the continue stallpoint (DTR\$K_STL_CONT) after each call. To continue, call DTR\$CONTINUE.

DTR\$CONTINUE

Return Status

SS\$_NORMAL

Call completed successfully.

DTR\$_BADHANDLE

The DAB is invalid.

DTR\$_BADNUMARG

Invalid number of arguments.

DTR\$_WRONGSTALL

Wrong stallpoint for this call. The stallpoint must be DTR\$K_STL_CONT, DTR\$K_STL_MSG, or DTR\$K_STL_LINE.

Other errors from RMS, system services, and Run-Time Library routines.

Examples

Create a Pascal procedure that prints out DATATRIEVE messages or print lines:

```
PROCEDURE PRINTSTUFF;  
  VAR STATUS : INTEGER;  
  BEGIN  
    WHILE (DAB.DAB$W_STATE = DTR$K_STL_MSG) OR  
          (DAB.DAB$W_STATE = DTR$K_STL_LINE) DO  
      BEGIN  
        WRITELN (MSG_BUFF);  
        STATUS := DTR$CONTINUE (DAB);  
      END  
    END;  
END;
```

Create a FORTRAN subroutine that suppresses the display of a number of DATATRIEVE messages but displays all other messages and print lines:

```
SUBROUTINE MESSAGE  
  INCLUDE 'DTR$LIBRARY:DAB'  
  INTEGER*4 RET_STATUS
```

```
C While DATATRIEVE is at either the message or print line  
C stallpoints.
```

```
      DO WHILE ((DAB$W_STATE .EQ. DTR$K_STL_MSG) .OR.  
1(DAB$W_STATE .EQ. DTR$K_STL_LINE))
```

```
C If DATATRIEVE is at the message stallpoint
```

```
      IF ((DAB$W_STATE .EQ. DTR$K_STL_MSG) .AND.
```

```
C and the message is one of the following:
```


DTR\$CONTINUE

```
1 ((DAB$L_CONDITION .EQ. %LOC(DTR$_RECFFOUND)) .OR.  
2 (DAB$L_CONDITION .EQ. %LOC(DTR$_ASSUMELIT)) .OR.  
3 (DAB$L_CONDITION .EQ. %LOC(DTR$_NONDIGIT)) .OR.  
4 (DAB$L_CONDITION .EQ. %LOC(DTR$_SUCCESS))) THEN
```

C Use the call DTR\$CONTINUE to skip the message.

```
    RET_STATUS = DTR$CONTINUE (DAB)  
    ELSE
```

C Otherwise, print out the message buffer.

```
    WRITE (6, *) MSG_BUFF(1:DAB$W_MSG_LEN)
```

C Instruct DATATRIEVE to continue.

```
    RET_STATUS = DTR$CONTINUE (DAB)  
    END IF  
    END DO  
    RETURN  
40  END
```

DTR\$CREATE_UDK

DTR\$CREATE_UDK

Lets you add a keyword to DATATRIEVE. Commands and statements you add are called user-defined keywords (UDKs).

Format

DTR\$CREATE_UDK (dab, keyword-name, index, context)

Arguments

dab

Data type: data block
Access: read/write
Mechanism: by reference
Is the DATATRIEVE Access Block.

keyword-name

Data type: character string
Access: read-only
Mechanism: by descriptor
Is the name of the keyword.

index

Data type: word integer
Access: read-only
Mechanism: by reference
Is a number between 1 and 32767. It is the number that is returned in the DAB field DAB\$W_UDK_INDEX as the index of this keyword.

context

Data type: word integer
Access: read-only
Mechanism: by reference
Is the context DATATRIEVE uses to interpret the keyword. Table 3–1 lists the user-defined keyword contexts and their values.

DTR\$CREATE_UDK

Table 3–1 Types of Context for User-Defined Keywords

Context	Value	DATATRIEVE interprets the keyword as:
DTR\$K_UDK_SET	1	A SET command
DTR\$K_UDK_SET_NO	2	A SET NO command
DTR\$K_UDK_SHOW	3	A SHOW command
DTR\$K_UDK_STATEMENT	4	A statement
DTR\$K_UDK_COMMAND	5	A command

Usage Notes

- You can use this call when DATATRIEVE is at the command stallpoint (DTR\$K_STL_CMD).
- After you call DTR\$CREATE_UDK to create a keyword, you can call DTR\$DTR or DTR\$WINDOWS to get access to interactive DATATRIEVE. If you set the DTR\$DTR or DTR\$WINDOWS option DTR\$M_OPT_UDK, your program gets control when a user enters the keyword you have created.
- When a user enters your keyword, DATATRIEVE stalls at the user-defined keyword stallpoint (DTR\$K_STL_UDK). Your program can now perform the functions that make up the keyword.
- If your keyword includes arguments, you must use the call DTR\$GET_STRING to parse them.
- To finish processing of your keyword and continue from the user-defined keyword stallpoint (DTR\$K_STL_UDK), use the call DTR\$END_UDK.
- If the UDK context is DTR\$K_UDK_STATEMENT, you must pass one statement or a series of statements within a BEGIN-END block to DATATRIEVE . You should pass only one statement or BEGIN-END block for each time the user enters a UDK statement. Refer to Appendix A for a sample FORTRAN program (the program CORRELATE) showing how to define a UDK statement.
After you pass the statement, DATATRIEVE enters the end user-defined keyword stallpoint (DTR\$K_STL_END_UDK).
- You cannot use zero as an index. Digital reserves indexes -1 through -32768 for DATATRIEVE keywords.

DTR\$CREATE_UDK

Return Status

SS\$_NORMAL

Call completed successfully.

DTR\$_BADHANDLE

The DAB is invalid.

DTR\$_BADNUMARG

Invalid number of arguments. There must be four.

DTR\$_BADSTRDES

Invalid descriptor for keyword name.

DTR\$_BADUDKCTX

Invalid UDK context.

DTR\$_BADUDKIDX

Invalid index number. The index must be a positive number.

DTR\$_WRONGSTALL

Wrong stallpoint for this call. You must be at a DTR\$K_STL_CMD stallpoint

Other errors from RMS, system services, and Run-Time Library routines.

Example

Enable the interactive user to use several DCL commands in DATATRIEVE:

```
! PROGRAM: UDK
100 %INCLUDE "DTR$LIBRARY:DAB.BAS"
! Declare the initialization and terminal server calls
! as functions.
EXTERNAL INTEGER FUNCTION DTR$INIT, DTR$DTR
! Declare the exit and normal status.
EXTERNAL LONG CONSTANT DTR$_EXIT, SS$_NORMAL
DECLARE INTEGER INIT_OPTIONS, DTR_OPTIONS, RET_STATUS
! Assign the initial options.
INIT_OPTIONS = DTR$K_SEMI_COLON_OPT &
OR DTR$K_UNQUOTED_LIT &
OR DTR$K_FORMS_ENABLE
! Initialize the Interface.
```

DTR\$CREATE_UDK

```
500  RET_STATUS = DTR$INIT (DAB BY REF, 100% BY REF, MSG_BUFF, &
      AUX_BUFF, INIT_OPTIONS BY REF)
      ! Check to see if DATATRIEVE was initialized.
      IF RET_STATUS <> SS$NORMAL THEN
          PRINT "DATATRIEVE initialization failed."
          GOTO 7000
      ! Create the user-defined keywords.
      ! UDK number 1 = CLEAR_SCREEN
      !               2 = DIRECTORY
      !               3 = MAIL
      !               4 = SPAWN
      !               5 = SHOW_UDKS
1000  RET_STATUS = DTR$CREATE_UDK (DAB BY REF, 'CLEAR_SCREEN', 1% BY REF, &
      DTR$K_UDK_COMMAND BY REF)
      RET_STATUS = DTR$CREATE_UDK (DAB BY REF, 'DIRECTORY', 2% BY REF, &
      DTR$K_UDK_COMMAND BY REF)
      RET_STATUS = DTR$CREATE_UDK (DAB BY REF, 'MAIL', 3% BY REF, &
      DTR$K_UDK_COMMAND BY REF)
      RET_STATUS = DTR$CREATE_UDK (DAB BY REF, 'SPAWN', 4% BY REF, &
      DTR$K_UDK_COMMAND BY REF)
      RET_STATUS = DTR$CREATE_UDK (DAB BY REF, 'UDKS', 5% BY REF, &
      DTR$K_UDK_SHOW BY REF)
      ! Declare the options for the DTR$DTR call.
2000  DTR_OPTIONS = DTR$M_OPT_UDK ! Return to program on a UDK &
      OR DTR$M_OPT_CONTROL_C      &
      OR DTR$M_OPT_STARTUP        &
      OR DTR$M_OPT_FOREIGN        &
      OR DTR$M_OPT_BANNER
      ! Call the terminal server.
2500  RET_STATUS = DTR$DTR (DAB BY REF, DTR_OPTIONS BY REF)
      ! Check for EXIT or CTRL/Z.
      GOTO 6000 IF RET_STATUS = DTR$_EXIT
      ON DAB$_UDK_INDEX           &
      GOSUB 3100, 3200, 3300, 3400, 3500
      GOTO 2500
      ! UDK 1 - User entered CLEAR_SCREEN.
3100  CALL LIB$ERASE_PAGE (1% BY REF, 1% BY REF)
      ! End the UDK.
```

DTR\$CREATE_UDK

```
RET_STATUS = DTR$END_UDK BY REF (DAB)
RETURN

! UDK 2 - User entered DIRECTORY.
3200 RET_STATUS = LIB$SPAWN ("DIRECTORY")
RET_STATUS = DTR$END_UDK BY REF (DAB)
RETURN

! UDK 3 - User entered MAIL.
3300 RET_STATUS = LIB$SPAWN ("MAIL")
RET_STATUS = DTR$END_UDK BY REF (DAB)
RETURN

! UDK 4 - User entered SPAWN.
3400 RET_STATUS = LIB$SPAWN ()
RET_STATUS = DTR$END_UDK BY REF (DAB)
RETURN

! UDK 5 - User entered SHOW UDKS.
3500 PRINT " "
PRINT " User-Defined Keywords Available"
PRINT " "
PRINT " CLEAR_SCREEN - clears the screen"
PRINT " DIRECTORY - displays files in the default directory"
PRINT " SPAWN - creates a subprocess"
PRINT " MAIL - invokes VMS MAIL"

RET_STATUS = DTR$END_UDK BY REF (DAB)
RETURN

6000 RET_STATUS = DTR$FINISH BY REF (DAB)
7000 END
```

This program shows you how to add commands to DATATRIEVE by using Run-Time Library routines. Refer to Appendix A for an example of how to add statements in a FORTRAN program (see the program CORRELATE).

Note that the DTR\$M_OPT_FOREIGN option to the DTR\$DTR or DTR\$WINDOWS call lets you execute commands and statements from DCL level. For example:

```
$ UDK := $DB0:[KELLER]UDK.EXE
$ UDK READY PERSONNEL; CLEAR_SCREEN; PRINT FIRST 5 PERSONNEL
```

DTR\$DTR

DTR\$DTR

Invokes the DATATRIEVE terminal server. Your program gets access to all of the DATATRIEVE interactive data management capabilities. Users of your program cannot tell that they are running a program and not interactive DATATRIEVE .

Use this call for access to the DATATRIEVE terminal server. For access to the DATATRIEVE DECwindows terminal server see the DTR\$WINDOWS call.

Format

DTR\$DTR (dab, options-code)

Arguments

dab

Data type: data block

Access: read/write

Mechanism: by reference

Is the DATATRIEVE Access Block.

options-code

Data type: longword integer

Access: read-only

Mechanism: by reference

Is a bit mask of options you select, where each bit in the options-code argument identifies a separate option. The default for options-code is 0.

The following tables describe the DTR\$DTR options and list their values. Table 3-2 describes the DTR\$DTR options that determine when the DATATRIEVE terminal server returns control to your program. Table 3-3 describes the DTR\$DTR options that enable various DATATRIEVE terminal server functions.

Table 3-2 DTR\$DTR Control Options

Option	Value	Terminal server returns control to your program when:
DTR\$M_OPT_CMD	1	Stallpoint is DTR\$K_STL_CMD

(continued on next page)

DTR\$DTR

Table 3–2 (Cont.) DTR\$DTR Control Options

Option	Value	Terminal server returns control to your program when:
DTR\$M_OPT_PRMPPT	2	Stallpoint is DTR\$K_STL_PRMPPT
DTR\$M_OPT_LINE	4	Stallpoint is DTR\$K_STL_LINE
DTR\$M_OPT_MSG	8	Stallpoint is DTR\$K_STL_MSG
DTR\$M_OPT_PGET	16	Stallpoint is DTR\$K_STL_PGET
DTR\$M_OPT_PPUT	32	Stallpoint is DTR\$K_STL_PPUT
DTR\$M_OPT_CONT	64	Stallpoint is DTR\$K_STL_CONT
DTR\$M_OPT_UDK	128	Stallpoint is DTR\$K_STL_UDK
DTR\$M_OPT_DTR_UDK	256	User enters a DATATRIEVE keyword
DTR\$M_OPT_END_UDK	512	Stallpoint is DTR\$K_STL_END_UDK
DTR\$M_OPT_UNWIND	1024	Condition is DTR\$_UNWIND—user enters CTRL/C or CTRL/Z during execution of a command or statement

Table 3–3 DTR\$DTR Terminal Server Options

Option	Value	This option enables:
DTR\$M_OPT_CONTROL_C	2048	DATATRIEVE CTRL/C handling
DTR\$M_OPT_STARTUP	4096	Execution of the startup command file the first time you call DTR\$DTR with this option
DTR\$M_OPT_FOREIGN	8192	Execution of the foreign command line
DTR\$M_OPT_BANNER	16384	Display of the DATATRIEVE login banner the first time you call DTR\$DTR with this option
DTR\$M_OPT_REMOVE_CTLC	32768	Removal of CTRL/C handling when leaving the DTR\$DTR call and returning to the program

(continued on next page)

DTR\$DTR

Table 3–3 (Cont.) DTR\$DTR Terminal Server Options

Option	Value	This option enables:
DTR\$M_OPT_KEYDEFS	65536	Definition of keypad keys as specified in the file associated with the logical name DTR\$KEYDEFS the first time you call DTR\$DTR with this option

Usage Notes

- The DATATRIEVE terminal server cannot be active at the same time as the DATATRIEVE DECwindows terminal server. If you have already made a call to DTR\$WINDOWS, you cannot make a call to DTR\$DTR until you have called DTR\$FINISH_WINDOWS.
- The DATATRIEVE DECwindows terminal server cannot be active at the same time as the DATATRIEVE terminal server. If you have already made a call to DTR\$DTR, you cannot make a call to DTR\$WINDOWS until you have called DTR\$FINISH.
- You can use the options-code argument to specify the stallpoints you want the terminal server to handle and the stallpoints at which you want your program to take control. For example, you can call DTR\$DTR to have DATATRIEVE display messages and print lines and then return control to your program by specifying DTR\$M_OPT_CMD in the options-code argument.
- If your program gives control to the terminal server and the user ends the DATATRIEVE session by entering CTRL/Z or EXIT in response to the DTR> prompt, control returns to your program.
- If you specify the option DTR\$M_OPT_UNWIND, DATATRIEVE returns control to your program whenever it detects an unwind condition. An unwind condition occurs when the user enters CTRL/C or CTRL/Z, or when DATATRIEVE encounters a DTR\$UNWIND call in your program.

The DTR\$M_OPT_CONTROL_C causes DATATRIEVE to turn on CTRL/C handling so that a CTRL/C is not interpreted as a CTRL/Y. The terminal server handles the CTRL/C breaks.

DTR\$DTR

Note

A BASIC program that calls DATATRIEVE cannot have the CTRL/C trapping enabled by BASIC while DATATRIEVE is executing. Disable CTRL/C trapping in your program with the RCTLC function before the program calls DATATRIEVE. If the BASIC CTRLC function is enabled while DATATRIEVE is executing, pressing CTRL/C can produce unexpected results.

- DTR\$DTR assigns a channel number to the DAB\$W_TT_CHANNEL DAB field if a number has not been previously assigned. DATATRIEVE uses the value of DAB\$W_TT_CHANNEL as the input/output channel for ADT, TDMS forms, Guide Mode, and help. (For information about deassigning channels, see Chapter 2.)
- DTR\$DTR calls the Run-Time Library Screen Management Facility (SMG) to create a key definition table and two virtual keyboards if the values have not been previously assigned. DATATRIEVE stores the key table identifier in DAB\$L_KEYTABLE_ID, the virtual keyboard identifier for command input in DAB\$L_COMMAND_KEYBOARD, and the virtual keyboard identifier for input from prompting expressions in DAB\$L_PROMPT_KEYBOARD.
- You can use DTR\$DTR with the call DTR\$CREATE_UDK to add your own keywords to interactive DATATRIEVE (see the discussion of DTR\$CREATE_UDK in this chapter).
- You can specify the option DTR\$M_OPT_STARTUP to invoke a startup command file pointed to by the logical DTR\$STARTUP. This command file is executed only the first time you call DTR\$DTR with the DTR\$M_OPT_STARTUP option set.

Return Status

SS\$_NORMAL

Call completed successfully.

DTR\$_BADNUMARG

Invalid number of arguments.

DTR\$_BADSTALL

Invalid stallpoint in the DAB.

DTR\$_EXIT

User entered either EXIT or CTRL/Z.

DTR\$DTR

DTR\$_NOSMG

The DECwindows terminal server is active.

DTR\$_UNWIND

Program called DTR\$UNWIND or user entered CTRL/C or CTRL/Z to terminate DATATRIEVE statement.

Other errors from RMS, system services, and Run-Time Library routines.

Examples

Use DTR\$DTR to handle error messages in a FORTRAN program:

```
IF (DAB$L_CONDITION .NE. %LOC(DTR$_SUCCESS))
1   RET_STATUS = DTR$DTR (DAB, DTR$M_OPT_CMD)
```

Use DTR\$DTR to display print lines in a COBOL program:

```
IF DTR$K_STL_LINE THEN
   RET_STATUS = "DTR$DTR" USING DAB DTR$M_OPT_CMD.
```

Use DTR\$DTR in a BASIC program to simulate interactive DATATRIEVE.

Note that the option DTR\$M_OPT_STARTUP executes a DATATRIEVE startup command file, if you have one.

```
100   %INCLUDE "DTR$LIBRARY:DAB"

      ! Declare the initialization and terminal server calls as
      ! functions.

      EXTERNAL INTEGER FUNCTION DTR$INIT, DTR$DTR

      DECLARE INTEGER DTR_OPTIONS, RET_STATUS

      ! Declare the normal and exit status.

      EXTERNAL INTEGER CONSTANT DTR$_BADNUMARG, &
                                DTR$_BADSTALL, &
                                DTR$_EXIT, &
                                SS$_NORMAL

      ! Initialize the interface.

500   RET_STATUS = DTR$INIT (DAB BY REF, 100% BY REF, MSG_BUFF, &
                           AUX_BUFF, DTR$K_SEMI_COLON_OPT BY REF )

      ! Check to see if DATATRIEVE was initialized.

      IF RET_STATUS <> SS$_NORMAL THEN
         PRINT "DATATRIEVE initialization failed."
         GOTO 2000

      ! Set options include commands to: execute a startup file,
      ! enable CTRL/C handling, allow invocation command lines,
      ! and display a startup banner.
```

DTR\$DTR

```
DTR_OPTIONS      =      DTR$M_OPT_CONTROL_C  &
                   OR DTR$M_OPT_STARTUP    &
                   OR DTR$M_OPT_FOREIGN    &
                   OR DTR$M_OPT_BANNER

! Call the terminal server.
1000  RET_STATUS = DTR$DTR (DAB BY REF, DTR_OPTIONS BY REF)
! Check the status.
SELECT RET_STATUS
CASE SS$NORMAL
    PRINT "Check your DTR$DTR options."
    PRINT "You returned control to the program."
CASE DTR$_EXIT
    PRINT "Bye."
CASE ELSE
    CALL LIB$SIGNAL (RET_STATUS BY VALUE)
END SELECT
1500  RET_STATUS = DTR$FINISH BY REF (DAB)
2000  END
```

You can use this program as the framework for customizing interactive DATATRIEVE. Refer to Appendix A for more examples of how to use DTR\$DTR.

DTR\$END_UDK

DTR\$END_UDK

Ends processing of a user-defined keyword.

Format

DTR\$END_UDK (dab)

Argument

dab

Data type: data block

Access: read/write

Mechanism: by reference

Is the DATATRIEVE Access Block.

Usage Notes

- To use this call you must be at the command, user-defined keyword, or end user-defined keyword stallpoint (DTR\$K_STL_CMD, DTR\$K_STL_UDK, or DTR\$K_STL_END_UDK).
- If your UDK is a DATATRIEVE statement, pass a statement or a series of statements within a BEGIN-END block to DATATRIEVE. When you finish passing the statement, DATATRIEVE enters the end user-defined keyword stallpoint (DTR\$K_STL_END_UDK). To continue, you must use the call DTR\$END_UDK. Refer to Appendix A for a sample FORTRAN program (program CORRELATE) that shows how to define and end a UDK statement.

Return Status

SS\$ _NORMAL

Call completed successfully.

DTR\$ _BADHANDLE

The DAB is invalid.

DTR\$ _BADNUMARG

Invalid number of arguments. There must be one.

DTR\$ _WRONGSTALL

Wrong stallpoint for this call. You must be at the DTR\$K_STL_CMD, DTR\$K_STL_UDK, or DTR\$K_STL_END_UDK stallpoint.

DTR\$END_UDK

Other errors from RMS, system services, and Run-Time Library routines.

Example

End processing of a user-defined keyword in a FORTRAN program:

```
      .  
      .  
      .  
C Create the UDK.  
      RET_STATUS = DTR$CREATE_UDK (DAB, 'SPAWN', 1, DTR$K_UDK_COMMAND)  
C Select DTR$DTR options.  
      DTR_OPTS = DTR$M_OPT_CONTROL_C ! Enable Control C handling  
1      + DTR$M_OPT_STARTUP ! Execute startup command file  
2      + DTR$M_OPT_FOREIGN ! Execute invocation command lines  
3      + DTR$M_OPT_BANNER ! Display DATATRIEVE banner  
4      + DTR$M_OPT_UDK ! Return on DTR$K_STL_UDK  
C Call the terminal server.  
100     RET_STATUS = DTR$DTR (DAB, DTR_OPTS)  
      ! Finish on CTRL/Z or EXIT.  
      GOTO 999 IF RET_STATUS = DTR$_EXIT  
C When user enters SPAWN, the program gets control.  
C The stallpoint is DTR$K_STL_UDK.  
      IF (DAB$W_UDK_INDEX .EQ. 1) THEN  
C Use a Run-Time Library routine to spawn a subprocess.  
      RET_STATUS = LIB$SPAWN()  
      END IF  
C Terminate processing of the UDK.  
      RET_STATUS = DTR$END_UDK (DAB)  
C Recall the terminal server.  
      GO TO 100  
999     RET_STATUS = DTR$FINISH (DAB)  
      END
```

DTR\$FINISH

DTR\$FINISH

Ends a program's interaction with the DATATRIEVE terminal server invoked by the DTR\$DTR call.

To end a program's interaction with the DATATRIEVE DECwindows terminal server invoked by the DTR\$WINDOWS call, see the DTR\$FINISH_WINDOWS call.

Format

DTR\$FINISH (dab)

Argument

dab

Data type: data block

Access: read/write

Mechanism: by reference

Is the DATATRIEVE Access Block.

Usage Notes

- DTR\$FINISH is similar to the DATATRIEVE exit command. DTR\$FINISH finishes all DATATRIEVE sources, releases all collections, tables, and variables, and ends the DATATRIEVE Call Interface session.
- After your program closes the DATATRIEVE Call Interface with a call to DTR\$FINISH, the program must reinitialize the DAB before it can be used in another call to DTR\$INIT. That is, the values of all the fields must be set to zero.

Return Status

SS\$_NORMAL

Call completed successfully.

DTR\$_BADHANDLE

The DAB is invalid.

DTR\$_BADNUMARG

Invalid number of arguments.

DTR\$FINISH

DTR\$_NOSMG

The DECwindows terminal server is active.

Other errors from RMS, system services, and Run-Time Library routines.

Example

Close the DATATRIEVE session in a FORTRAN program:

```
C Close the DATATRIEVE Interface.  
   STATUS = DTR$FINISH (DAB)
```


DTR\$FINISH_WINDOWS

DTR\$FINISH_WINDOWS

Ends a program's interaction with the DATATRIEVE DECwindows terminal server invoked by DTR\$WINDOWS.

To end a session with the DATATRIEVE terminal server, see the DTR\$FINISH call.

Format

DTR\$FINISH_WINDOWS (dab)

Argument

dab

Data type: data block

Access: read/write

Mechanism: by reference

Is the DATATRIEVE Access Block.

Usage Notes

- Use DTR\$FINISH_WINDOWS to end an interactive DATATRIEVE session with the DATATRIEVE DECwindows interface invoked with the DTR\$WINDOWS call. Use DTR\$FINISH to end a session initiated by a call to DTR\$DTR.
- DTR\$FINISH_WINDOWS is similar to the DATATRIEVE exit command. DTR\$FINISH_WINDOWS finishes all DATATRIEVE sources, releases all collections, tables, and variables, and ends the DATATRIEVE Call Interface session.
- After your program closes the DATATRIEVE Call Interface with a call to DTR\$FINISH_WINDOWS, the program must reinitialize the DAB before it can be used in another call to DTR\$INIT. That is, the values of all the fields must be set to zero.

Return Status

SS\$_NORMAL

Call completed successfully.

DTR\$_BADHANDLE

The DAB is invalid.

DTR\$FINISH_WINDOWS

DTR\$_BADNUMARG

Invalid number of arguments.

DTR\$_NOWINDOWS

User tried to call DTR\$FINISH_WINDOWS without first invoking the DATATRIEVE DECwindows interface with a DTR\$WINDOWS call.

Other errors from RMS, system services, and Run-Time Library routines.

Example

Close the DATATRIEVE session in a FORTRAN program:

```
C Close the DATATRIEVE DECwindows Interface.  
   STATUS = DTR$FINISH_WINDOWS (DAB)
```

DTR\$GET_PORT

DTR\$GET_PORT

Gets a record from DATATRIEVE and passes it to your program.

Format

DTR\$GET_PORT (dab, record-buffer)

Arguments

dab

Data type: data block

Access: read/write

Mechanism: by reference

Is the DATATRIEVE Access Block.

record-buffer

Data type: data block

Access: write

Mechanism: by reference

Is a buffer to hold the record. The record buffer must be large enough to contain all of the fields defined in the port.

Usage Notes

- To transfer records from DATATRIEVE to your program, you must define a record buffer in your program to receive the records. For example, here is a record buffer for YACHTS in a FORTRAN program:

```
CHARACTER*1 YACHT(41)
CHARACTER*10 BUILDER
CHARACTER*10 MODEL
CHARACTER*6 RIG
CHARACTER*3 LOA
CHARACTER*5 DISP
CHARACTER*2 BEAM
CHARACTER*5 PRICE
EQUIVALENCE (YACHT(1), BUILDER),
1 (YACHT(11), MODEL),
2 (YACHT(21), RIG),
3 (YACHT(27), LOA),
4 (YACHT(30), DISP),
5 (YACHT(35), BEAM),
6 (YACHT(37), PRICE)
```

DTR\$GET_PORT

- If you write your programs in a high-level language that supports VAX CDD/Repository (such as BASIC, COBOL, and FORTRAN), you need not code the entire record buffer into your program. You can copy a record definition from the dictionary and use it as a record buffer.

Be sure that your record definition does not contain a field name that is a reserved word in the programming language you are using. For example, the YACHT record definition contains a field named TYPE, which is a COBOL reserved word. You should change this field name if you want to copy the record into a COBOL program. For example:

```
RECORD YACHT_COBOL USING
01 BOAT.
   03 BOAT_TYPE.
       06 MANUFACTURER PIC X(10)
           QUERY_NAME IS BUILDER.
       .
       .
       .
;
```

- To transfer records to your program, you must also define a port. A port is a single record buffer that can be referenced by DATATRIEVE and your program. All transfer of records between DATATRIEVE and the host program is done through ports. To define a port, use the DEFINE PORT command or the DECLARE PORT statement.
- The DEFINE PORT command inserts your port definition into the dictionary. On the DEFINE PORT command line you specify a name for the port and an associated record definition in the following format:

```
DEFINE PORT path-name [USING] record-path-name;
```

To define a port for YACHTS records, you can use the following command in interactive DATATRIEVE:

```
DTR> DEFINE PORT YPORT USING YACHT;
```

You can also use the DTR\$COMMAND call to pass this command to DATATRIEVE from your program. If you use DEFINE PORT to create a port, ready the port for write access before using it.

- The DECLARE PORT statement creates a temporary port with the name you specify and readies the port for write access. DATATRIEVE does not enter a definition of the port in the dictionary. You can issue the DECLARE PORT statement only from your program. You must define a record in the statement.

DTR\$GET_PORT

Here is an example of the DECLARE PORT statement in a BASIC program:

```
STATUS = DTR$COMMAND (DAB BY REF, "DECLARE PORT PT1 & 01 PTYPE PIC X(20).;")
```

- To pass data from DATATRIEVE to your program, you must first tell DATATRIEVE to store a record in a port. If you use the STORE statement to store data in a port, include the USING clause to specify the fields you want to store, as in the following examples:

```
FOR YACHTS WITH LOA GT 30 STORE YPORT USING BUILDER = BUILDER  
FOR YACHTS STORE PT1 USING PT1_TYPE = TYPE
```

You also can use an Assignment statement in the following format:

```
port-name = rse
```

An example of an Assignment statement that stores data into a port is:

```
YPORT = YACHTS WITH LOA GT 30
```

If you want to use this Assignment statement to store only some fields of a DATATRIEVE record into a port, define your port to include only the fields you want to store.

- A DATATRIEVE statement that stores records into a port results in the get port stallpoint (DTR\$K_STL_PGET). When you are at this stallpoint, you can use the call DTR\$GET_PORT to copy a record from a port to a record buffer in your program. You can use the call DTR\$GET_PORT only when DATATRIEVE is at the get port stallpoint (DTR\$K_STL_PGET).
- After the DTR\$GET_PORT call, DATATRIEVE is usually at the get port or message stallpoint (DTR\$K_STL_PGET or DTR\$K_STL_MSG). If DATATRIEVE has more records to store into the port, the stallpoint is the get port stallpoint.
- The message buffer contains the name of the port when DATATRIEVE reaches the get port stallpoint (DTR\$K_STL_PGET).
- Be careful when you use record buffers that are the same size as the ports; if the sizes do not match, you may cause DATATRIEVE to write over the addresses following the record buffer, causing errors in your program.

Return Status

SS\$ NORMAL

Call completed successfully.

DTR\$ BADHANDLE

The DAB is invalid.

DTR\$GET_PORT

DTR\$_BADNUMARG

Invalid number of arguments.

DTR\$_BADSTALL

Invalid stallpoint in the DAB.

DTR\$_WRONGSTALL

Wrong stallpoint for this call. You must be at the get port stallpoint (DTR\$_STL_PGET).

Other errors from RMS, system services, and Run-Time Library routines.

Example

Set up a record buffer and use a port in a COBOL program:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.    SEEYACHTS.
*****
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
DATA DIVISION.
WORKING-STORAGE SECTION.

*****
* Define a record buffer by copying a dictionary record      *
* definition.                                               *
*****
        COPY "CDD$TOP.COBOL.YACHT_COBOL" FROM DICTIONARY.

*****
* Copy in the DAB and set up program variables.            *
*****
COPY "DTR$LIBRARY:DAB.LIB".

01 STACK_SIZE PIC 99 COMP VALUE 100.

01 CONT PIC X.
01 BOOL PIC X(30).
01 COMMAND_LINE PIC X(80).
01 RET_STATUS PIC S9(9) COMP.

PROCEDURE DIVISION.
```

DTR\$GET_PORT

```
010-SHOW-YACHTS.
  MOVE "Y" TO CONT.
  PERFORM 050-INITIALIZE-INTERFACE.
  PERFORM 100-READY-DOMAINS-AND-PORT.
  PERFORM 200-GET-BOOL UNTIL
    CONT EQUAL "N" OR CONT EQUAL "n".
  PERFORM 999-EOJ.

050-INITIALIZE-INTERFACE.
*****
* Initialize the Interface. *
*****

  RET_STATUS = "DTR$INIT" USING DAB STACK_SIZE
              BY DESCRIPTOR MSG_BUFF AUX_BUFF.

*****
* Use DTR$COMMAND to define a port. *
*****

  MOVE "DEFINE PORT YPORT USING YACHT_COBOL;"
    TO COMMAND_LINE.
  RET_STATUS = "DTR$COMMAND" USING DAB
              BY DESCRIPTOR COMMAND_LINE.

*****
* Use the call DTR$DTR to print out any print lines or error *
* messages. Set the DTR$M_OPT_CMD option so that DATATRIEVE *
* returns control to the program on the DTR$K_STL_CMD *
* stallpoint. *
*****

  RET_STATUS = "DTR$DTR" USING DAB DTR$M_OPT_CMD.

100-READY-DOMAINS-AND-PORT.
  MOVE "READY YACHTS; READY YPORT WRITE;"
    TO COMMAND_LINE.
  RET_STATUS = "DTR$COMMAND" USING DAB
              BY DESCRIPTOR COMMAND_LINE.

*****
* Have DATATRIEVE display messages and return control. *
*****

  RET_STATUS = "DTR$DTR" USING DAB DTR$M_OPT_CMD.

200-GET-BOOL.
  DISPLAY "ENTER A BOOLEAN EXPRESSION, SUCH AS LOA > 30.".
  ACCEPT BOOL.

*****
* Command DATATRIEVE to store the port. After this call, *
* DATATRIEVE stalls at the DTR$K_STL_PGET stallpoint. *
*****
```

DTR\$GET_PORT

```
MOVE "YPORT = YACHTS WITH !CMD;"
      TO COMMAND_LINE.
RET_STATUS = "DTR$COMMAND" USING DAB
      BY DESCRIPTOR COMMAND_LINE BOOL.

*****
* Check for error messages. Print column headers. *
*****

IF DTR$K_STL_MSG
RET_STATUS = "DTR$DTR" USING DAB DTR$M_OPT_CMD
      GO TO 200-GET-BOOL
ELSE
      DISPLAY " "
      DISPLAY "BUILDER" " " "MODEL" " " "RIG" " "
      "LOA" " " "BEAM" " " "WEIGHT" " " "PRICE"
      DISPLAY " ".
      PERFORM 300-PASS-AND-DISPLAY-RECORDS UNTIL NOT DTR$K_STL_PGET.
      PERFORM 400-DO-MORE.

300-PASS-AND-DISPLAY-RECORDS.

*****
* DATATRIEVE is at the DTR$K_STL_PGET stallpoint. Use the call *
* DTR$GET_PORT to copy a record from YPORT to the record buffer.*
*****
      RET_STATUS = "DTR$GET_PORT" USING DAB BOAT.
      DISPLAY MANUFACTURER " " MODEL " " RIG " "
      LENGTH_OVER_ALL " " BEAM " " DISPLACEMENT " " PRICE.

400-DO-MORE.
      RET_STATUS = "DTR$DTR" USING DAB DTR$M_OPT_CMD.
      DISPLAY " ".
      DISPLAY "ENTER Y TO SEE MORE RECORDS, N TO QUIT."
      ACCEPT CONT.

999-EOJ.
      RET_STATUS = "DTR$FINISH" USING DAB.
      STOP RUN.
```


DTR\$GET_STRING

DTR\$GET_STRING

Gets a string from DATATRIEVE and passes it to your program. You can use this call in your program to interpret the arguments to a user-defined keyword.

Format

DTR\$GET_STRING (dab, token-type, [string], [length], [compare-string])

Arguments

dab

Data type: data block
Access: read/write
Mechanism: by reference
Is the DATATRIEVE Access Block.

token-type

Data type: longword integer
Access: read-only
Mechanism: by reference
Is the type of token you want DATATRIEVE to return.

A token can be a character string delimited by spaces or any other identifiable unit within the command string. For example, in the following command line, the word OPEN and the file specification DUA0:[SAMPLE]DTR.LOG are both tokens:

```
OPEN DUA0:[SAMPLE]DTR.LOG
```

You can also return the entire command string by selecting the DTR\$K_TOK_COMMAND type. Table 3–4 lists all the token types you can specify.

Table 3–4 Token Types for the DTR\$GET_STRING Call

Token Type	Value	Action
DTR\$K_TOK_TOKEN	1	Retrieves a single token
DTR\$K_TOK_FILENAME	2	Retrieves a file specification
DTR\$K_TOK_PICTURE	3	Retrieves a picture string

(continued on next page)

DTR\$GET_STRING

Table 3–4 (Cont.) Token Types for the DTR\$GET_STRING Call

Token Type	Value	Action
DTR\$K_TOK_COMMAND	4	Retrieves the remainder of the string
DTR\$K_TOK_TEST_TOKEN	5	Retrieves a single token if it matches the compare string and has the specified length
DTR\$K_TOK_LIST_ELEMENT	6	Retrieves all tokens until the next comma or the end of the line
DTR\$K_TOK_TEST_EOL	7	Checks for the end of the string and returns either DTR\$_MORESTR, if there are any remaining tokens, or DTR\$_ENDOFSTR, if there are no more tokens

Arguments

string

Data type: character string

Access: write

Mechanism: by descriptor

Is the retrieved string. This parameter is required for all token types except DTR\$K_TOK_TEST_EOL.

length

Data type: word integer

Access: read/write

Mechanism: by reference

Is the length of the string. If the type is DTR\$K_TOK_TEST_TOKEN, you can use length to determine whether or not the token was read.

compare-string

Data type: character string

Access: read-only

Mechanism: by descriptor

Is a string that should be matched by the token in the string argument.

DTR\$GET_STRING

This argument is examined only for type DTR\$K_TOK_TEST_TOKEN. For other types, it is ignored. If the type is DTR\$K_TOK_TEST_TOKEN and this descriptor is not specified, the result of the DTR\$GET_STRING call is the same as for type DTR\$K_TOK_TOKEN.

Usage Notes

- You can use this call only when DATATRIEVE is at the user-defined keyword stallpoint (DTR\$K_STL_UDK).
- After this call, DATATRIEVE remains at the user-defined keyword stallpoint (DTR\$K_STL_UDK).
- If you use DTR\$GET_STRING to parse a number of arguments to a UDK, you may need to check the status of your call frequently. The status codes DTR\$_ENDOFSTR, DTR\$_MORESTR, and DTR\$_TRUNCSTR are useful in parsing the command string the user enters.

For example, suppose you create a UDK, CORRELATE, that lets users enter an argument string:

```
DTR> CORRELATE LOA, BEAM
```

You have set the DTR\$M_OPT_UDK option to DTR\$WINDOWS or to DTR\$DTR so that when the user enters CORRELATE, your program gets control. You can check if any strings have been entered after CORRELATE as follows:

```
RET_STATUS = DTR$GET_STRING (DAB, DTR$K_TOK_TOKEN,  
1      FIELD1)  
IF      (RET_STATUS .EQ. %LOC(DTR$_ENDOFSTR))  
THEN WRITE (6,*) 'Enter a field name: '
```

If a field name has been entered, you can check for a comma with the DTR\$K_TOK_TEST_TOKEN type:

```
IF (RET_STATUS .EQ. %LOC(DTR$_MORESTR)) THEN  
  RET_STATUS = DTR$GET_STRING (DAB, DTR$K_TOK_TEST_TOKEN,  
1      COMMA, 1, ',')  
  IF (RET_STATUS .EQ. %LOC(DTR$_ENDOFSTR) ) THEN  
    WRITE (6,*) 'Enter a comma or the next field name: "  
  END IF  
END IF
```

Refer to Appendix A for a sample FORTRAN program (see the subroutine PARSE) showing how to parse a string entered after a user-defined keyword.

DTR\$GET_STRING

Return Status

SS\$_NORMAL

Call completed successfully.

DTR\$_BADHANDLE

The DAB is invalid.

DTR\$_BADNUMARG

Invalid number of arguments. You must specify at least three.

DTR\$_BADSTRDES

Invalid string descriptor.

DTR\$_BADTOKTYP

Invalid token type.

DTR\$_ENDOFSTR

There are no more tokens.

DTR\$_MORESTR

There are more valid tokens.

DTR\$_TRUNCSTR

The token is too long for the string. The required token length is specified in the length argument. Another GET_STRING call will retrieve the same token.

DTR\$_WRONGSTALL

Wrong stallpoint for this call. The stallpoint must be DTR\$K_STL_UDK.

Other errors from RMS, system services, and Run-Time Library routines.

Examples

You have a DATATRIEVE procedure that produces a long report and sends it to a file:

```
PROCEDURE SALARY_REPORT
READY PERSONNEL SHARED
REPORT PERSONNEL ON DB0:[DIETTERICH]MONTHLY.REP
.
.
.
END_REPORT
END_PROCEDURE
```

DTR\$GET_STRING

You want to submit this procedure for batch processing and continue your session with DATATRIEVE. Create the UDK SUBMIT in a BASIC program:

```
100  %INCLUDE "DTR$LIBRARY:DAB"
      ! Declare the initialization and terminal server
      ! calls as functions.
      EXTERNAL INTEGER FUNCTION DTR$INIT, DTR$DTR
      ! Declare the success and exit status.
      EXTERNAL LONG CONSTANT SS$NORMAL, DTR$_EXIT
      DECLARE INTEGER DTR_OPTIONS, INIT_OPTIONS, RET_STATUS
      DECLARE STRING DTR_PROCEDURE
      ! Assign the initial options.
      INIT_OPTIONS = DTR$K_SEMI_COLON_OPT &
                   OR DTR$K_UNQUOTED_LIT &
                   OR DTR$K_FORMS_ENABLE
      ! Initialize the Interface.
500  RET_STATUS = DTR$INIT (DAB BY REF, 100% BY REF, MSG_BUFF, &
                          AUX_BUFF, INIT_OPTIONS BY REF )
      ! Check to see if DATATRIEVE was initialized.
      IF RET_STATUS <> SS$NORMAL THEN
          PRINT "DATATRIEVE initialization failed."
          GOTO 6000
      ! Create the user defined keyword.
1000 RET_STATUS = DTR$CREATE_UDK (DAB BY REF, 'SUBMIT', 1% BY REF, &
                                DTR$K_UDK_COMMAND BY REF)
      ! Declare the options for the DTR$DTR call.
2000 DTR_OPTIONS = DTR$_OPT_UDK ! Return to program on UDK &
                  + DTR$_OPT_CONTROL_C &
                  + DTR$_OPT_STARTUP &
                  + DTR$_OPT_FOREIGN &
                  + DTR$_OPT_BANNER
      ! Call the terminal server.
2500 RET_STATUS = DTR$DTR (DAB BY REF, DTR_OPTIONS BY REF)
      ! Check for EXIT or CTRL/Z.
      GOTO 6000 IF RET_STATUS = DTR$_EXIT
3000 OPEN "TEMP.COM" FOR OUTPUT AS FILE #2%
      ! Use DTR$GET_STRING to get the procedure name.
```

DTR\$GET_STRING

```
RET_STATUS = DTR$GET_STRING (DAB BY REF, DTR$K_TOK_COMMAND, &
                           DTR_PROCEDURE BY DESC)

! User entered SUBMIT; read in a procedure name.
IF DTR_PROCEDURE = "" THEN
INPUT "Enter procedure name: "; DTR_PROCEDURE

! Write a command file that executes the DATATRIEVE procedure.
4000 PRINT #2%, "$ RUN SYS$SYSTEM:DTR32"
      PRINT #2%, "EXECUTE ";DTR_PROCEDURE
      PRINT #2%, "$ EXIT"
      CLOSE 2%

! Submit the command file.
RET_STATUS = LIB$SPAWN ("SUBMIT/QUEUE=SYS$BATCH/NOPRINTER "+ &
                       "TEMP/NOTIFY/DELETE/LOG=[]")

! End the UDK.
RET_STATUS = DTR$END_UDK BY REF (DAB)

! Recall the terminal server.
GO TO 2500

6000 RET_STATUS = DTR$FINISH BY REF ( DAB )
      END
```

DTR\$INFO

DTR\$INFO

Returns information about the DATATRIEVE object you specify. To use this call, you pass in:

- A number that identifies the object (the object-id)
- An option code that specifies what kind of information you want (the info-code)
- In some cases, a number that specifies which element in a series you want information about (the index)

Depending on the option code you select, DTR\$INFO returns the following:

- An identifying number for another object, such as the object-id of a second or third readied domain
- A string, such as one of the lines in a query header
- The length of the string returned

Format

DTR\$INFO (dab, object-id, info-code, ret-val, [output-string], [index])

Arguments

dab

Data type: data block

Access: read/write

Mechanism: by reference

Is the DATATRIEVE Access Block.

object-id

Data type: longword integer

Access: read-only

Mechanism: by reference

Is a number that identifies a DATATRIEVE object. You obtain this number with the DTR\$LOOKUP or DTR\$INFO call.

info-code

Data type: byte integer

Access: read-only

Mechanism: by reference

DTR\$INFO

Is a number that identifies the type of information you want. The info-code options you can select are declared in inclusion files provided in DTR\$LIBRARY as part of the DATATRIEVE installation kit. Table 3-5 lists the valid options.

ret-val

Data type: longword integer

Access: write

Mechanism: by reference

Is a number DATATRIEVE returns to your program. If you specify the output-string argument, then ret-val is the length of the output string. If you do not use the output-string argument, then ret-val is either the object-id or a number that provides your program with information. (See Table 3-5.)

output-string

Data type: character string

Access: write

Mechanism: by descriptor

Is a string DATATRIEVE uses to return the information you request. Table 3-5 shows when you can use this optional argument and what information it returns.

index

Data type: longword integer

Access: read-only

Mechanism: by value

Is a number that specifies which of a series of elements you want to access. For example, if you want to retrieve the third line in a multiline statement, then index should equal three.

Table 3-5 lists the following:

- The info-code options you can select
- The number of arguments you must pass for each option
- The information DTR\$INFO returns for each option

DTR\$INFO

Table 3–5 Info-Code Options

Info-code options	Number of arguments required	Information DTR\$INFO returns
Object-id of domains, statements, collections, and subschemas		
Pass in a GLV identifier obtained with the DTR\$LOOKUP call		
DTR\$K_INF_GLV_FIRST_DOM	4	Object-id of first domain in readied domain list
DTR\$K_INF_GLV_FIRST_COL	4	Object-id of first collection in collection list
DTR\$K_INF_GLV_FIRST_SSC	4	Object-id of first open subschema
DTR\$K_INF_GLV_STA_OBJ	4	Object-id of last statement entered
DTR\$K_INF_GLV_DEF_DIC	5	Default dictionary path string
Information about statements		
Pass in an object-id obtained with DTR\$K_INF_GLV_STA_OBJ		
DTR\$K_INF_GLV_STA_CNT	4	Number of lines in statement
DTR\$K_INF_GLV_STA_LINE	6	Line specified by index in output-string
Information about domains		
Pass in an object-id obtained with DTR\$K_INF_GLV_FIRST_DOM		
DTR\$K_INF_DOM_FLD	4	Object-id of top-level field
DTR\$K_INF_DOM_FORM	4	Object-id of form associated with domain

(continued on next page)

DTR\$INFO

Table 3-5 (Cont.) Info-Code Options

Info-code options	Number of arguments required	Information DTR\$INFO returns
Information about domains		
DTR\$K_INF_DOM_SHARE	4	Number specifying domain access option: EXCLUSIVE = 1, SHARED = 2, PROTECTED = 3
DTR\$K_INF_DOM_ACCESS	4	Number specifying domain access mode: READ = 1, WRITE = 2, MODIFY = 3, EXTEND = 4
DTR\$K_INF_DOM_NAME	5	Name of domain in output-string
DTR\$K_INF_DOM_NEXT_DOM	4	Object-id of next domain in readied domain list
DTR\$K_INF_DOM_SSC	4	Object-id of subschema of VAX DBMS domain
DTR\$K_INF_DOM_REC_LEN	4	Length of record associated with current domain
Information about fields		
Pass in the object-id of a top-level field obtained with DTR\$K_INF_DOM_FLD		
DTR\$K_INF_FLD_NAME	5	Name of field in output-string
DTR\$K_INF_FLD_QNAME	5	Query name of field in output-string
DTR\$K_INF_FLD_QHDR	4	Object-id of query header
DTR\$K_INF_FLD_PICTURE	5	Picture string for field in output-string
DTR\$K_INF_FLD_EDIT	5	Edit string for field in output-string

(continued on next page)

DTR\$INFO

Table 3–5 (Cont.) Info-Code Options

Info-code options	Number of arguments required	Information DTR\$INFO returns
Information about fields		
DTR\$K_INF_FLD_DTYPE	4	Number of VAX data type for field
DTR\$K_INF_FLD_OFFSET	4	Offset of field within record in bytes
DTR\$K_INF_FLD_LENGTH	4	Length of field in bytes
DTR\$K_INF_FLD_SCALE	4	Scale factor of field
DTR\$K_INF_FLD_CHILD	6	Object-id of child of field specified in object-id; index number specifies which child
DTR\$K_INF_FLD_CNT	4	Count of children of field specified in object-id
DTR\$K_INF_FLD_LIST	4	Zero if not top-level field in a list (field defined with OCCURS clause)
DTR\$K_INF_FLD_REDEFINES	4	Object-id of field defined with REDEFINES clause
DTR\$K_INF_FLD_VIRTUAL	4	Nonzero if field is a COMPUTED BY field; else, value is zero
DTR\$K_INF_FLD_FILLER	4	Zero if not field FILLER; else, a number greater than zero
DTR\$K_INF_FLD_MISSING	5	Descriptor of the MISSING VALUE

(continued on next page)

DTR\$INFO

Table 3–5 (Cont.) Info-Code Options

Info-code options	Number of arguments required	Information DTR\$INFO returns
Information about fields		
DTR\$K_INF_FLD_MISSING_TXT	5	Descriptor of the MISSING VALUE text for a string field
DTR\$K_INF_FLD_SEG_STRING	4	One if the field is a relational database segmented string field; else, zero.
Information about collections		
Pass in the object-id of a collection obtained with DTR\$K_INF_GLV_FIRST_COL		
DTR\$K_INF_COL_FLD	4	Object-id of top-level field
DTR\$K_INF_COL_DROPPED	4	One if a selected record was dropped; else, zero
DTR\$K_INF_COL_ERASED	4	One if a selected record was erased; else, zero
DTR\$K_INF_COL_INVISIBLE	4	One if a collection is invisible (see Usage Notes); else, zero
DTR\$K_INF_COL_NAME	5	Name of collection in output-string
DTR\$K_INF_COL_NEXT_COL	4	Object-id of next collection in collection list

(continued on next page)

DTR\$INFO

Table 3–5 (Cont.) Info-Code Options

Info-code options	Number of arguments required	Information DTR\$INFO returns
Information about forms		
Pass in the object-id of a form obtained with DTR\$K_INF_DOM_FORM		
DTR\$K_INF_FRM_NAME	5	Name of form in output-string
DTR\$K_INF_FRM_LIBRARY	5	Name of form library in output-string
Information about VAX DBMS subschemas, sets, and domains		
Pass in the object-id of a subschema obtained with DTR\$K_INF_GLV_FIRST_SSC		
DTR\$K_INF_SSC_NAME	5	Name of subschema in output-string
DTR\$K_INF_SSC_SET	4	Address of last active set; null if no sets active
DTR\$K_INF_SSC_NEXT	4	Object-id of previous active subschema
Pass in the object-id of a set obtained with DTR\$K_INF_SSC_SET		
DTR\$K_INF_SET_NAME	5	Name of set in output-string
DTR\$K_INF_SET_NEXT	4	Object-id of previous active set
DTR\$K_INF_SET_SDP	4	Object-id of set/domain pair
DTR\$K_INF_SET_SINGULAR	4	One if set is singular (system owned); else, zero
Pass in the object-id of a domain/set pair obtained with DTR\$K_INF_SET_SDP		

(continued on next page)

DTR\$INFO

Table 3–5 (Cont.) Info-Code Options

Info-code options	Number of arguments required	Information DTR\$INFO returns
Information about VAX DBMS subschemas, sets, and domains		
DTR\$K_INF_SDP_NEXT	4	Object-id of next domain/set pair
DTR\$K_INF_SDP_DOMAIN	4	Object-id of domain associated with set
DTR\$K_INF_SDP_TENANCY	4	One if domain is member of current set
DTR\$K_INF_SDP_INSERT	4	Insertion type for member domains: Manual = 1, Automatic = 2
DTR\$K_INF_SDP_RETAIN	4	Retention type for member domains: Fixed = 1, Mandatory = 2, Optional = 3
Information about header lines		
Pass in the object-id of a header obtained with DTR\$K_INF_FLD_QHDR		
DTR\$K_INF_HDR_CNT	4	Number of lines in header
DTR\$K_INF_HDR_STRING	6	Header line specified by index in output-string

(continued on next page)

DTR\$INFO

Table 3–5 (Cont.) Info-Code Options

Info-code options	Number of arguments required	Information DTR\$INFO returns
Information about plots		
Pass in the object-id of a plot obtained with DTR\$LOOKUP call		
DTR\$K_INF_PLO_CNT	4	Number of plot arguments
DTR\$K_INF_PLO_PAI	6	Object-id of plot argument specified in index
DTR\$K_INF_HDR_STRING	6	Header line specified by index in output-string
Pass in the object-id of a plot argument obtained with DTR\$K_INF_PLO_PAI		
DTR\$K_INF_PAI_PROMPT	5	Prompting string for plot argument if one exists; length in ret-val = 0 if no string
DTR\$K_INF_PAI_DTYPE	4	Number of VAX data type expected for argument

Usage Notes

- To use this call you must copy the INFO inclusion file into your program. You can find INFO inclusion files in FORTRAN, COBOL, BASIC, Pascal, PL/I, and C in the DTR\$LIBRARY directory. The file name for each file is INFO and the file type identifies the language it is designed to be used with.
- You can use the DTR\$INFO call when DATATRIEVE is at the command, user-defined keyword, or end user-defined keyword stallpoint (DTR\$K_STL_CMD, DTR\$K_STL_UDK, or DTR\$K_STL_END_UDK).
- One of the info-code options, DTR\$K_INF_COL_INVISIBLE, allows you to check for “invisible” collections. An invisible collection is formed as follows:

DTR\$INFO

```
DTR> READY FAMILIES !FAMILIES is a hierarchical domain.
DTR> FIND FAMILIES !Form an unnamed collection of FAMILIES.
[14 records found]
DTR> SELECT 1

DTR> SHOW CURRENT
Collection CURRENT
  Domain: FAMILIES
  Number of Records: 14
  Selected Record: 1

DTR> FIND KIDS !KIDS is a list. Form an
                !unnamed collection of KIDS.
[2 records found]
DTR> SHOW CURRENT
Collection CURRENT
  Domain: FAMILIES
  Number of Records: 2
  No Selected Record
```

After you form the collection KIDS, the first collection of FAMILIES is invisible.

Return Status

SS\$_NORMAL

Call completed successfully.

DTR\$_BADHANDLE

The DAB is invalid.

DTR\$_BADNUMARG

Invalid number of arguments.

DTR\$_INFBADCOD

Invalid info-code.

DTR\$_INFBADID

Invalid object-id.

DTR\$_WRONGSTALL

Wrong stallpoint for this call. The stallpoint must be DTR\$K_STL_CMD, DTR\$K_STL_UDK, or DTR\$K_STL_END_UDK.

Other errors from RMS, system services, and Run-Time Library routines.

DTR\$INFO

Examples

Write a FORTRAN program to determine if a selected record in a collection has been dropped:

```
·  
·  
·
```

C Use DTR\$LOOKUP to get the global variable identifier.

```
RET_STATUS = DTR$LOOKUP (DAB, DTR$K_INF_TYPE_GLV, GLV_ID)
```

C Use DTR\$INFO with the DTR\$K_INF_GLV_FIRST_COL option to get the object-id of the first collection in the collection list.

```
RET_STATUS = DTR$INFO (DAB, GLV_ID, DTR$K_INF_GLV_FIRST_COL,  
1 COL_ID)
```

C Call DTR\$INFO again, using the object-id obtained with the last call.
C Specify the DTR\$K_INF_COL_NAME option to get the collection name.

```
RET_STATUS = DTR$INFO (DAB, COL_ID, DTR$K_INF_COL_NAME,  
1 LENGTH, COL_NAME)
```

C Call DTR\$INFO with the DTR\$K_INF_COL_DROPPED option to find out if
C the selected record has been dropped.

C The ret-val argument returns one if the record has been dropped,
C zero if the record has not been dropped.

```
RET_STATUS = DTR$INFO (DAB, COL_ID, DTR$K_INF_COL_DROPPED, DROPPED)  
  
IF (DROPPED .EQ. 1 ) THEN  
WRITE (6, *) 'Selected record from ', COL_NAME, 'dropped.'  
END IF  
  
RET_STATUS = DTR$FINISH (DAB)
```

Create a UDK that enables users to find out the maximum number of arguments required for a plot. For example:

```
DTR> PLOT_ARGUMENTS MULTI_BAR  
MULTI_BAR not found in the default plots directory  
DTR> SET PLOTS CDD$TOP.DTR$LIB.VT125  
DTR> PLOT_ARGUMENTS MULTI_BAR  
Maximum number of arguments for MULTI_BAR is 4  
DTR> PLOT_ARGUMENTS  
Enter plot name: WOMBAT  
Maximum number of arguments for WOMBAT is 0
```

DTR\$INFO

The following program creates the UDK PLOT_ARGUMENTS:

```
INCLUDE 'DTR$LIBRARY:DAB'
INCLUDE 'DTR$LIBRARY:INFO'

INTEGER*2      DTR_OPTS, PLOT_LEN, ARGUMENT_CNT
INTEGER*4      DTR$_EXIT, DTR$DTR, RET_STATUS, PLOT_ID
CHARACTER*15   PLOT_NAME
EXTERNAL      DTR$_EXIT

C Initialize the DATATRIEVE Call Interface.

    RET_STATUS = DTR$INIT (DAB, 100, MSG_BUFF, AUX_BUFF,
    1                      DTR$_EXIT)

C Create the UDK PLOT_ARGUMENTS.

    RET_STATUS = DTR$CREATE_UDK (DAB, 'PLOT_ARGUMENTS', 1, DTR$_EXIT)

C Declare terminal server call options and call DTR$DTR.

    DTR_OPTS = DTR$_OPT_CONTROL_C ! Enable Control C handling
    1      + DTR$_OPT_STARTUP      ! Execute startup command file
    2      + DTR$_OPT_FOREIGN      ! Execute invocation command lines
    3      + DTR$_OPT_BANNER       ! Display DATATRIEVE banner
    4      + DTR$_OPT_UDK          ! Return on UDK

10    RET_STATUS = DTR$DTR (DAB, DTR_OPTS)
    IF (RET_STATUS .EQ. %LOC(DTR$_EXIT)) THEN
    GO TO 999
    END IF

    DO WHILE ((DAB$_UDK_INDEX .EQ. 1) .AND.
    1 (DAB$_STATE .EQ. DTR$_STL_UDK))

C User entered PLOT_ARGUMENTS plot-name.
C Get the name of the plot.

        RET_STATUS = DTR$GET_STRING (DAB, DTR$_TOK_COMMAND,
    1                                PLOT_NAME, PLOT_LEN)

C User entered PLOT_ARGUMENTS. Prompt for a plot name.

        IF (PLOT_NAME .EQ. ' ') THEN
        WRITE (6, 20)
        20      FORMAT (X, 'Enter plot name: ', $)
        READ (5, 30) PLOT_LEN, PLOT_NAME
        30      FORMAT (Q, A)
        END IF

C Use DTR$LOOKUP to get the object-id for the specified plot.

        RET_STATUS = DTR$LOOKUP (DAB, DTR$_INF_TYPE_PLOT, PLOT_ID,
    1                                PLOT_NAME)

C If there is no such plot, inform the user.
```

DTR\$INFO

```
        IF (PLOT_ID .EQ. 0) THEN
          WRITE (6, 35) PLOT_NAME
35         FORMAT (1X, A<PLOT_LEN>, ' not found in the
          1 default plots directory')
          GO TO 50
        END IF

C Use DTR$INFO to get the maximum number of arguments for the plot.
        RET_STATUS = DTR$INFO (DAB, PLOT_ID, DTR$K_INF_PLO_CNT, ARGUMENT_CNT)
        WRITE (6,40) PLOT_NAME, ARGUMENT_CNT
40         FORMAT (' Maximum number of arguments for ', A<PLOT_LEN>,
          1 ' is ', I1)
50         RET_STATUS = DTR$END_UDK (DAB)
        END DO
        GO TO 10
999        RET_STATUS = DTR$FINISH(DAB)
        END
```

Create a UDK that enables users to display information about record definitions as follows:

```
DTR> READY OWNERS, PERSONNEL
DTR> SHOW RECORD_LENGTH
Domain: PERSONNEL
PERSON  <Group Field>
  ID    < 5 Bytes >
  EMPLOYEE_STATUS (STATUS) < 11 Bytes >
  EMPLOYEE_NAME (NAME) <Group Field>
    FIRST_NAME (F_NAME) < 10 Bytes >
    LAST_NAME (L_NAME) < 10 Bytes >
  DEPT  < 3 Bytes >
  START_DATE < 8 Bytes >
  SALARY < 5 Bytes >
  SUP_ID < 5 Bytes >
PERSONNEL has a total record length of 57 bytes
Domain: OWNERS
OWNER  <Group Field>
  NAME < 10 Bytes >
  BOAT_NAME < 17 Bytes >
  TYPE <Group Field>
    BUILDER < 10 Bytes >
    MODEL < 10 Bytes >
OWNERS has a total record length of 47 bytes
```

DTR\$INFO

The BASIC program that creates the UDK RECORD_LENGTH shows how you can use DTR\$INFO to get information about elementary fields in a field tree. Here is the program:

```
100    %INCLUDE "DTR$LIBRARY:DAB" !Copy in the DAB.
200    %INCLUDE "DTR$LIBRARY:INFO" !Copy in the INFO inclusion file.

! Declare initialization and terminal server calls as functions.
! Declare exit and normal status and call options.

EXTERNAL INTEGER FUNCTION DTR$INIT, DTR$DTR
EXTERNAL LONG CONSTANT DTR$_EXIT, SS$_NORMAL
DECLARE INTEGER INIT_OPTIONS, DTR_OPTIONS

250    REM    FN.GET.FIELDS Function                                &
!*****
! Define a recursive BASIC function to get information about
! fields in the record:
! FLD_ID%      = the object-id of a field
! FIELD_COUNT% = the number of children (subfields) a field has
! I%           = local variable used to pass number of the field
!              level
300    DEF FN.GET.FIELDS (FLD_ID%, FIELD_COUNT%, I%)

! Add one to the number of the field level.

LEVEL% = I% + 1%

! Determine the number of spaces to go in front of the field name.
! (3 spaces per level).

SPACE.FILLER$ = SPACE$(I% * 3)

! Find out if the field is a FILLER field.

RET_STATUS% = DTR$INFO (DAB BY REF, FLD_ID%, &
                      DTR$K_INF_FLD_FILLER, FILLER% , ,)

! If the field is a FILLER field, name it FILLER.

IF FILLER% = 1% &
  THEN
    FLD_NAME$ = "FILLER"
  ELSE

! If the field is not FILLER, get its name.

    RET_STATUS% = DTR$INFO (DAB BY REF, FLD_ID%, &
                          DTR$K_INF_FLD_NAME, NAME_LEN%, FLD_NAME$,)

! Get the query name of the field.

350    RET_STATUS% = DTR$INFO (DAB BY REF, FLD_ID%, &
                          DTR$K_INF_FLD_QNAME, QNAME.LEN%, QNAME$, )

! If the field has a query name, enclose it with parentheses.
```

DTR\$INFO

```
IF QNAME.LEN% <> 0% &
  THEN
    QNAME$ = "(" + QNAME$ + ")"
  ELSE
    QNAME$ = SPACE$(0%)

! If the field has no children (subfields),
! then it is an elementary field.
400 IF FIELD_COUNT% = 0% &
  ! Find the length (in bytes) of the elementary field.
  THEN RET_STATUS% = DTR$INFO (DAB BY REF, FLD_ID%, &
    DTR$K_INF_FLD_LENGTH, FLD_LENGTH% , ,)
  ! Print out field name, query name, and length.
  PRINT SPACE.FILLER$; FLD_NAME$; " "; QNAME$; &
    " <";FLD_LENGTH$;"Bytes >"

! If the field is not an elementary field,
! print out the field name with a notice that it is a group field.
  ELSE
    PRINT SPACE.FILLER$; FLD_NAME$; " "; QNAME$;" <Group Field>"
  ! Loop through the children of the group field.
  FOR I% = 1% TO FIELD_COUNT%
  ! Get the object-id of the child field.
    RET_STATUS% = DTR$INFO (DAB BY REF, FLD_ID%, &
      DTR$K_INF_FLD_CHILD, FLD_ID2%, ,I% BY VALUE)
  ! Find out if this new field has children.
    RET_STATUS% = DTR$INFO (DAB BY REF, FLD_ID2%, &
      DTR$K_INF_FLD_CNT, FIELD_COUNT2%, ,)
  ! Call this function (recurse), passing it the object-id,
  ! child count, and level number of the child field.
    A% = FN.GET.FIELDS (FLD_ID2%, FIELD_COUNT2%, LEVEL%)
  ! Do the same with next child.
    NEXT I%

! Subtract 1 from the current level to get back to previous
! level.
425 LEVEL% = LEVEL% - 1%
450 END DEF
460 REM      End of Function                                     &
!*****
```

DTR\$INFO

```
! Assign options and initialize the Interface.
475  INIT_OPTIONS = DTR$K_SEMI_COLON_OPT &
          + DTR$K_UNQUOTED_LIT &
          + DTR$K_FORMS_ENABLE
500  RET_STATUS% = DTR$INIT (DAB BY REF, 100%, MSG_BUFF, AUX_BUFF, &
          INIT_OPTIONS)

! Check to see if DATATRIEVE was initialized.
IF RET_STATUS% <> SS$NORMAL THEN
    PRINT "DATATRIEVE initialization failed."
    GOTO 8000

! Create the user-defined keyword RECORD_LENGTH.
1000 RET_STATUS% = DTR$CREATE_UDK (DAB BY REF, 'RECORD_LENGTH', 1%, &
          DTR$K_UDK_SHOW)

! Use DTR$LOOKUP to get the global variable identifier.
RET_STATUS% = DTR$LOOKUP (DAB BY REF, DTR$K_INF_TYPE_GLV, GLV_ID% ,)

! Declare the options for the DTR$DTR call.
2000 DTR_OPTIONS = DTR$M_OPT_UDK !Return to program on a UDK &
          + DTR$M_OPT_CONTROL_C &
          + DTR$M_OPT_STARTUP &
          + DTR$M_OPT_FOREIGN &
          + DTR$M_OPT_BANNER

! Call the terminal server.
3000 RET_STATUS% = DTR$DTR (DAB BY REF, DTR_OPTIONS)

! Execute the following loop until DTR$DTR returns
! DTR$_EXIT status.
UNTIL RET_STATUS% = DTR$_EXIT

! Get the object-id of the first domain.
RET_STATUS% = DTR$INFO (DAB BY REF, GLV_ID%, &
          DTR$K_INF_GLV_FIRST_DOM, DOM_ID%, ,)

! If no address is returned, there are no readied domains.
IF DOM_ID% = 0% &
    THEN PRINT "No ready domains"

! Execute the following loop until there are no more
! ready domains.
4000  UNTIL DOM_ID% = 0%

! Get the name of the domain and display it.
```

DTR\$INFO

```
RET_STATUS% = DTR$INFO (DAB BY REF, DOM_ID%, DTR$K_INF_DOM_NAME, &
                        DOM_NAME_LEN%, DOM_NAME$,)
PRINT
PRINT "Domain: "; DOM_NAME$
! Get the address of the first field in the domain.
RET_STATUS% = DTR$INFO (DAB BY REF, DOM_ID%, DTR$K_INF_DOM_FLD, &
                        FLD_ID% , ,)
! Find the number of elementary fields the first field has.
RET_STATUS% = DTR$INFO (DAB BY REF, FLD_ID%, DTR$K_INF_FLD_CNT, &
                        FIELD_COUNT%,,)
! Call the function to print out the fields and field lengths of the
! domain record definition.
A% = FN.GET.FIELDS(FLD_ID%, FIELD_COUNT%, 1%)
! Find the total length of the record.
RET_STATUS% = DTR$INFO (DAB BY REF, DOM_ID%, &
                        DTR$K_INF_DOM_REC_LEN, REC_LENGTH% , ,)
PRINT
IF REC_LENGTH% = 0% &
! If the record length is 0, then the domain is
! a view domain. Print out this information.
    THEN
        PRINT DOM_NAME$;" is a view domain and has no record length."
    ELSE
! Display the total length of the record.
        PRINT DOM_NAME$; " has a total record length of";
        PRINT REC_LENGTH%; "bytes"
! Get the object-id next domain.
5000   RET_STATUS% = DTR$INFO (DAB BY REF, DOM_ID%, &
                            DTR$K_INF_DOM_NEXT_DOM, DOM_ID% , ,)
NEXT
RET_STATUS% = DTR$END_UDK(DAB BY REF) !End the UDK.
! Recall the terminal server.
RET_STATUS% = DTR$DTR ( DAB BY REF, DTR_OPTIONS)
6000   NEXT
7000   RET_STATUS% = DTR$FINISH BY REF ( DAB )
8000   END
```

DTR\$INIT

DTR\$INIT

Initializes the DATATRIEVE Call Interface, letting your program and DATATRIEVE communicate.

Format

DTR\$INIT (dab, [size], [msg-buff], [aux-buff], [options-code])

Arguments

dab

Data type: data block
Access: read/write
Mechanism: by reference
Is the DATATRIEVE Access Block.

size

Data type: longword integer
Access: read-only
Mechanism: by reference
Is the number of virtual pages that the DATATRIEVE stack uses. Exhausting the DATATRIEVE stack causes an access violation and means that following DATATRIEVE calls cannot be completed. Stack size has no other effect on program performance. The default of 100 is sufficient for most users.
If you do not specify a stack size or if you specify a value less than 100, DATATRIEVE uses the default stack size of 100 pages.

msg-buff

Data type: character string
Access: read/write
Mechanism: by descriptor
Is the message buffer. You can omit this argument if you put the address and length of the message buffer in the DAB. If a character string is longer than 255 characters, the maximum the message buffer can contain, DATATRIEVE returns the error DTR\$_BADSTRDES.

aux-buff

Data type: character string
Access: read/write
Mechanism: by descriptor

DTR\$INIT

Is the auxiliary message buffer. You can omit this argument if you put the address and length of this buffer in the DAB. If a character string is longer than 255 characters, the maximum the message buffer can contain, DATATRIEVE returns the error DTR\$_BADSTRDES.

options-code

Data type: longword integer

Access: read-only

Mechanism: by reference

Is a bit mask of options you select, where each bit in the options-code argument identifies a separate option. The default for options-code is 0.

The options you can select and their values are as follows:

DTR\$K_SEMI_COLON_OPT = 1

Enables termination of DATATRIEVE commands and statements without the semicolon. If this option is not set, all DATATRIEVE commands and statements must end with a semicolon. Setting this option is equivalent to using the SET SEMICOLON command.

DTR\$K_UNQUOTED_LIT = 16

DATATRIEVE assumes a string is a literal if it cannot interpret the string as a valid field name.

DTR\$K_SYNTAX_PROMPT = 32

Enables DATATRIEVE syntax prompting. If your program passes an incomplete command or statement to DATATRIEVE, DATATRIEVE prompts for the continuation of the input line. Setting this option is equivalent to using the SET PROMPT command. SET PROMPT is the default.

DTR\$K_IMMED_RETURN = 64

Enables the continue stallpoint (DTR\$K_STL_CONT). DATATRIEVE returns control to your program immediately after each call.

DTR\$K_FORMS_ENABLE = 128

Enables the DATATRIEVE and Forms Interface. If your program either accesses domains that use a form or passes DISPLAY_FORM or WITH_FORM statements, and if the value in DAB\$W_TT_CHANNEL is different from zero, then DATATRIEVE displays the form on your terminal screen. Setting this option is equivalent to using the SET FORM command.

DTR\$K_VERIFY = 256

DTR\$INIT

Enables terminal display of the contents of command files invoked within DATATRIEVE. Setting this option is equivalent to using the SET VERIFY command. SET NO VERIFY is the default.

DTR\$K_CONTEXT_SEARCH = 2048

Enables the DATATRIEVE Context Searcher. Setting this option is equivalent to using the SET SEARCH command. By default, the Context Searcher is not enabled. (See the description of the SET SEARCH command in the *VAX DATATRIEVE Reference Manual* for more information on the Context Searcher.)

DTR\$K_HYPHEN_DISABLED = 4096

Disables continuation of strings with a hyphen. If you set this option, the hyphen is not interpreted as a continuation character in command strings.

DTR\$K_MORE_COMMANDS = 8192

Enables execution of multiple command lines. If your program is at the command stallpoint (DTR\$K_STL_CMD) and you set this option, you can call DTR\$COMMAND a number of times and still be at a command stallpoint. DATATRIEVE does not parse or execute the commands and statements passed to it when DTR\$K_MORE_COMMANDS is in effect. You can use only the call DTR\$COMMAND when you set this option.

Before the last command line you pass to DATATRIEVE, you can clear the option and call DTR\$COMMAND. DATATRIEVE then parses and executes all the commands and statements you passed. Refer to Appendix A for a sample FORTRAN program (the CORRELATE program) showing how to use this option.

DTR\$K_ABORT = 16384

Enables termination of the execution statements and command files with the ABORT statement. Setting this option is equivalent to using the SET ABORT command. SET NO ABORT is the default.

DTR\$K_LOCK_WAIT = 32768

Causes DATATRIEVE to continue to attempt to access a record that is currently locked. Setting this option is equivalent to using the SET LOCK_WAIT command. SET NO LOCK_WAIT is the default.

DTR\$INIT

Usage Notes

- If the call is successful, DATATRIEVE enters the command stallpoint (DTR\$K_STL_CMD), the message buffer contains the DTR> prompt, and DAB\$L_CONDITION is set to zero. When DTR\$INIT initializes the DAB, it does the following:
 - Clears all the fields except for:
 - DAB\$W_TT_CHANNEL
 - DAB\$L_COMMAND_KEYBOARD
 - DAB\$L_PROMPT_KEYBOARD
 - DAB\$L_KEYTABLE_ID
 - Stores values supplied by the user.
 - Initializes some fields.
- To specify more than one option in the options-code argument, you can either add together the values for all the options you want, or you can use logical operators to create the bit mask. For example, the following BASIC statements produce the same value for the options-code argument:

```
OPTIONS_CODE = DTR$K_SEMI_COLON_OPT ! Semicolon is optional &
               + DTR$K_UNQUOTED_LIT ! Use unquoted literals &
               + DTR$K_FORMS_ENABLE ! Display forms

OPTIONS_CODE = DTR$K_SEMI_COLON_OPT ! Semicolon is optional &
               OR DTR$K_UNQUOTED_LIT ! Use unquoted literals &
               OR DTR$K_FORMS_ENABLE ! Display forms
```

The value of the options-code argument is stored in the DAB\$L_OPTIONS field of the DAB.

- Typically, you should make only one call to DTR\$INIT during a DATATRIEVE session.
- DTR\$INIT does not invoke the DATATRIEVE command startup file. See the call DTR\$DTR or DTR\$WINDOWS for the option that invokes the startup file.
- Your program can start up to five DATATRIEVE sessions at one time. Each session requires a separate Access Block and a call to DTR\$INIT to start the session. If you want to initialize DATATRIEVE more than five times, you must call DTR\$FINISH or DTR\$FINISH_WINDOWS to close one of the existing sessions, then call DTR\$INIT again to start the new session.

DTR\$INIT

- If you attempt to store values in one of the DAB fields that will be cleared, you will lose them. This happens because the DTR\$INIT call automatically clears several DAB fields before starting to use it.

Return Status

SS\$_NORMAL

Call completed successfully.

DTR\$_BADNUMARG

Invalid number of arguments. You must specify at least the DAB.

DTR\$_BADSTRDES

Invalid string descriptor in argument list.

DTR\$_USESLOEXH

Five concurrent DATATRIEVE streams already initialized. You cannot initialize another.

Other errors from RMS, system services, and Run-Time Library routines.

Examples

Initialize the DATATRIEVE Call Interface in a FORTRAN program. Check the status of the DTR\$INIT call:

C Get the definition of the DAB from the inclusion file.

```
INCLUDE 'DTR$LIBRARY:DAB'
```

C Declare variables for status checking.

```
INTEGER*4    INIT_OPTIONS
INTEGER*4    DTR$INIT
INTEGER*4    RET_STATUS
EXTERNAL     SS$_NORMAL
```

C Set options.

```
INIT_OPTIONS =
1      + DTR$K_SEMI_COLON_OPT      ! Semicolon is optional
2      + DTR$K_UNQUOTED_LIT       ! Use unquoted literals
3      + DTR$K_FORMS_ENABLE        ! Display forms
```

C Initialize the session with DATATRIEVE.

```
RET_STATUS = DTR$INIT (DAB,
1              100,
2              MSG_BUFF,
3              AUX_BUFF,
4              INIT_OPTIONS)
```

C Verify that the call was completed successfully.

DTR\$INIT

```
IF (RET_STATUS .NE. %LOC(SS$NORMAL)) THEN
    WRITE (6, *) ' DATATRIEVE initialization failed.'
    STOP
END IF
```

Initialize the DATATRIEVE Call Interface in a COBOL program. Check the status of the DTR\$INIT call:

```
.
.
.
COPY "DTR$LIBRARY:DAB.LIB".

01 STACK_SIZE    PIC 99    COMP VALUE 100.
01 INIT_OPTIONS  PIC 999   COMP.
01 RET_STATUS    PIC 9(9)  COMP.

.
.
.

ADD DTR$K_SEMI_COLON_OPT DTR$K_UNQUOTED_LIT
   DTR$K_FORMS_ENABLE GIVING INIT_OPTIONS.

CALL "DTR$INIT" USING DAB STACK_SIZE
   BY DESCRIPTOR MSG_BUFF AUX_BUFF
   BY REFERENCE INIT_OPTIONS
   GIVING RET_STATUS.

IF RET_STATUS IS FAILURE THEN
    DISPLAY "Initialization of DATATRIEVE failed."
    GO TO 999-BAD-INIT.
```

Initialize the DATATRIEVE Call Interface in a BASIC program. Check the status of the DTR\$INIT call:

```
100  %INCLUDE "DTR$LIBRARY:DAB.BAS"

EXTERNAL LONG CONSTANT SS$NORMAL

INIT_OPTIONS = DTR$K_SEMI_COLON_OPT &
               + DTR$K_UNQUOTED_LIT &
               + DTR$K_FORMS_ENABLE

RET_STATUS% = DTR$INIT ( DAB BY REF, 100% BY REF, MSG_BUFF, &
                       AUX_BUFF, INIT_OPTIONS BY REF )

! Check to see if DATATRIEVE was initialized.

IF RET_STATUS% <> SS$NORMAL THEN
    PRINT "DATATRIEVE initialization failed."
    GOTO 2000
```

DTR\$INIT

Initialize the DATATRIEVE Call Interface in a Pascal program. Note that the DTR\$INIT call is declared as an external procedure in the Pascal DAB:

```
%INCLUDE 'DTR$LIBRARY:DAB.PAS'

VAR      INIT_OPTIONS      : INTEGER;
         RET_STATUS        : INTEGER;

BEGIN

INIT_OPTIONS := DTR$K_SEMI_COLON_OPT
               + DTR$K_UNQUOTED_LIT
               + DTR$K_FORMS_ENABLE;

RET_STATUS   := DTR$INIT (DAB,
1              100,
2              MSG_BUFF,
3              AUX_BUFF,
4              INIT_OPTIONS);
```

DTR\$LOOKUP

DTR\$LOOKUP

Returns a number that identifies a DATATRIEVE object. You can use this call to:

- Find out if a dictionary object or DATATRIEVE keyword exists.
- Get an identifying number that you can use in the DTR\$INFO call. DTR\$INFO enables you to get information about domains, collections, record definitions, fields, subschemas, sets, and plots.

Format

DTR\$LOOKUP (dab, object-type, object-id, [object-name])

Arguments

dab

Data type: data block

Access: read/write

Mechanism: by reference

Is the DATATRIEVE Access Block.

object-type

Data type: byte integer

Access: read-only

Mechanism: by reference

Is the type of object that you want information about.

The object types you can select and their values are as follows:

DTR\$K_INF_TYPE_DOMAIN = 1

DTR\$LOOKUP returns the object-id of the domain specified in the object-name argument. The object-id is zero if the specified domain is not ready or not found.

DTR\$K_INF_TYPE_COLLECTION = 2

DTR\$LOOKUP returns the object-id of the collection specified in the object-name argument. The object-id is zero if the collection is not found.

DTR\$K_INF_TYPE_KEYWORD = 3

DTR\$LOOKUP

DTR\$LOOKUP returns a number not equal to zero if the string specified in the object-name argument is a DATATRIEVE keyword and zero if the string is not a keyword.

```
DTR$K_INF_TYPE_DIC_NAME = 4
```

DTR\$LOOKUP returns a number not equal to zero if the string specified in the object-name argument is a dictionary object and zero if the string is not a dictionary object. DTR\$LOOKUP uses your default dictionary directory if you do not specify a full path name in the object-name.

```
DTR$K_INF_TYPE_GLV = 5
```

DTR\$LOOKUP returns a global variable identifier (GLV). You can use this identifier in the DTR\$INFO call to get information about dictionary objects such as domains, collections, and fields. There is one GLV for each time you initialize DATATRIEVE, so you should not specify the object-name argument if you use this type.

```
DTR$K_INF_TYPE_PLOT = 6
```

DTR\$LOOKUP returns the object-id of the plot you pass in with the object-name argument. The object-id is zero if the plot is not found.

object-id

Data type: longword integer

Access: write

Mechanism: by reference

Is a number that identifies the object you want information about.

DTR\$LOOKUP returns the object-id to your program. The object-id is zero if the object you specify is not found.

object-name

Data type: character string

Access: read-only

Mechanism: by descriptor

Is the name of the object you want information about. DTR\$LOOKUP ignores this argument if the object type is DTR\$K_INF_TYPE_GLV.

DTR\$LOOKUP

Usage Notes

- If you use this call, you should copy the INFO inclusion file into your program. You can find INFO inclusion files in FORTRAN, COBOL, BASIC, PL/1, C and Pascal in the DTR\$LIBRARY directory.
- You can use this call when DATATRIEVE is at the command, user-defined keyword, or end user-defined keyword stallpoint (DTR\$K_STL_CMD, DTR\$K_STL_UDK, or DTR\$K_STL_END_UDK).
- If you specify the type DTR\$K_INF_TYPE_DOMAIN, DTR\$LOOKUP returns the object-id of readied domains only.
- You can use the types DTR\$K_INF_TYPE_KEYWORD and DTR\$K_INF_TYPE_DIC_NAME to check for names that duplicate keywords or existing dictionary path names. ADT uses these types in DTR\$LOOKUP calls.
- You can use the type DTR\$K_INF_TYPE_GLV to get identifiers of readied domains, collections, subschemas, sets, fields, and plots. (See the description of the DTR\$INFO call.)

Return Status

SS\$ _NORMAL

Call completed successfully.

DTR\$ _BADHANDLE

The DAB is invalid.

DTR\$ _BADNUMARG

Invalid number of arguments.

DTR\$ _INFNOTFOU

The object you specified in object-name was not found.

DTR\$ _WRONGSTALL

Wrong stallpoint for this call. The stallpoint must be DTR\$K_STL_CMD, DTR\$K_STL_UDK, or DTR\$K_STL_END_UDK.

Other errors from RMS, system services, and Run-Time Library routines.

DTR\$LOOKUP

Examples

In a BASIC program, verify that a string is a keyword or a synonym for a keyword:

```
.  
. .  
CALL DTR$LOOKUP (DAB BY REF, DTR$K_INF_TYPE_KEYWORD, &  
  ID, TEST_STRING BY DESC)  
SELECT ID  
CASE 0  
PRINT TEST_STRING; " is not a keyword."  
CASE ELSE  
PRINT TEST_STRING; " is a keyword."  
END SELECT
```

In a FORTRAN program, use the return status DTR\$_INFNOTFOU to verify that a specific dictionary directory exists:

```
EXTERNAL      DTR$_INFNOTFOU, DTR$LOOKUP  
INTEGER*4     DTR$LOOKUP, RET_STATUS  
. . .  
RET_STATUS = DTR$LOOKUP (DAB, DTR$K_INF_TYPE_DIC_NAME  
  CDD_OBJECT_ID, 'CDD$TOP.DTR$USERS.WOMBAT')  
IF (RET_STATUS .EQ. %LOC(DTR$_INFNOTFOU)) THEN  
  WRITE (6,*) 'CDD directory not found.'
```

For more examples of DTR\$LOOKUP, see the examples for the DTR\$INFO call.

DTR\$PORT_EOF

DTR\$PORT_EOF

Terminates passing of records from your program to DATATRIEVE.

Format

DTR\$PORT_EOF (dab)

Argument

dab

Data type: data block

Access: read/write

Mechanism: by reference

Is the DATATRIEVE Access Block.

Usage Note

You can use this call only if DATATRIEVE is at the put port stallpoint (DTR\$K_STL_PPUT).

Return Status

SS\$_NORMAL

Call completed successfully.

DTR\$_BADHANDLE

The DAB is invalid.

DTR\$_WRONGSTALL

Wrong stallpoint for this call. You must be at the DTR\$K_STL_PPUT stallpoint.

Other errors from RMS, system services, and Run-Time Library routines.

Examples

Use the calls DTR\$PUT_PORT and DTR\$PORT_EOF to store records in the domain YACHTS from a FORTRAN program:

DTR\$PORT_EOF

```
C***** PROGRAM: PUTPORT *****
C Include definition of the DAB.

    INCLUDE 'DTR$LIBRARY:DAB'

C Set up a buffer for records. Because input is from the terminal,
C declare the entire buffer and each field as CHARACTER types.

    CHARACTER*1 YACHT(41)
    CHARACTER*10 BUILDER
    CHARACTER*10 MODEL
    CHARACTER*6 RIG
    CHARACTER*3 LOA
    CHARACTER*5 DISP
    CHARACTER*2 BEAM
    CHARACTER*5 PRICE
    EQUIVALENCE (YACHT(1), BUILDER),
    1            (YACHT(11), MODEL),
    2            (YACHT(21), RIG),
    3            (YACHT(27), LOA),
    4            (YACHT(30), DISP),
    5            (YACHT(35), BEAM),
    6            (YACHT(37), PRICE)

    INTEGER*4 DTR$INIT
    INTEGER*4 DTR_OPTIONS
    INTEGER   RET_STATUS
    EXTERNAL  SS$_NORMAL

C Select DTR$DTR options.

    DTR_OPTIONS =
    1            DTR$M_OPT_CMD           ! Return on DTR$K_STL_CMD.
    2            + DTR$M_OPT_PPUT       ! Return on DTR$K_STL_PPUT.

C Initialize the session with DATATRIEVE.

    RET_STATUS = DTR$INIT (DAB, 100, MSG_BUFF, AUX_BUFF,
    1 DTR$K_SEMI_COLON_OPT)

C Verify that initialization was successful.

    IF (RET_STATUS .NE. %LOC(SS$_NORMAL)) THEN
        WRITE (6, *) ' DATATRIEVE initialization failed.'
        STOP
    END IF

C Ready domain.

    RET_STATUS = DTR$COMMAND (DAB, 'READY YACHTS WRITE')

C Display messages.

    RET_STATUS = DTR$DTR (DAB, DTR$M_OPT_CMD)

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

DTR\$PORT_EOF

```
RET_STATUS = DTR$COMMAND(DAB, 'DECLARE PORT BOAT_PORT USING
1 01 YACHT.
2   03 BOAT.
3     06 BUILDER PIC X(10).
4     06 MODEL PIC X(10).
5     06 RIG PIC X(6).
6     06 LOA PIC X(3).
7     06 DISP PIC X(5).
8     06 BEAM PIC XX.
9     06 PRICE PIC X(5).;')
```

C Display messages.

```
RET_STATUS = DTR$DTR (DAB, DTR$M_OPT_CMD)
```

C Read in a record. Note that the order your program reads records
C need not be the same as the order DATATRIEVE uses.

```
5   WRITE (6, 10)
10  FORMAT (' Enter BUILDER: ', $)
    READ (5, 100, END = 200) BUILDER

    WRITE (6, 20)
20  FORMAT (' Enter MODEL: ', $)
    READ (5, 100, END = 200) MODEL

    WRITE (6, 30)
30  FORMAT (' Enter RIG: ', $)
    READ (5, 100, END = 200) RIG

    WRITE (6, 40)
40  FORMAT (' Enter LENGTH: ', $)
    READ (5, 100, END = 200) LOA

    WRITE (6, 50)
50  FORMAT (' Enter BEAM: ', $)
    READ (5, 100, END = 200) BEAM

    WRITE (6, 60)
60  FORMAT (' Enter WEIGHT: ', $)
    READ (5, 100, END = 200) DISP

    WRITE (6, 70)
70  FORMAT (' Enter PRICE: ', $)
    READ (5, 100, END = 200) PRICE
100 FORMAT (A)
```

C Command DATATRIEVE to store YACHTS. DATATRIEVE stalls at
C the DTR\$K_STL_PPUT stallpoint.

```
RET_STATUS = DTR$COMMAND (DAB, 'FOR BOAT_PORT STORE YACHTS
1           USING BOAT = BOAT;')
```

C Use DTR\$PUT_PORT to pass the complete record.

```
RET_STATUS = DTR$PUT_PORT (DAB, %REF(YACHT))
```

DTR\$PORT_EOF

C Make sure record was stored; signal if it was not stored.

```
IF (DAB$W_STATE .EQ. DTR$K_STL_MSG)
1 RET_STATUS = DTR$DTR (DAB, DTR_OPTS)
```

C Inquire if user wishes to continue.

```
200 WRITE (6, 210)
210 FORMAT (' Do you wish to continue? ', $)
READ(5, 100)ANS
IF ((ANS .EQ. 'Y') .OR. (ANS .EQ. 'y')) THEN
GO TO 5
END IF
```

C Use the DTR\$PORT_EOF call to stop storing.

```
300 RET_STATUS = DTR$PORT_EOF (DAB)
RET_STATUS = DTR$DTR (DAB, DTR$M_OPT_CMD)
```

C End the Interface.

```
RET_STATUS = DTR$FINISH (DAB)
END
```

DTR\$PRINT_DAB

Displays the contents of the DAB.

Format

DTR\$PRINT_DAB (dab)

Argument

dab

Data type: data block

Access: read/write

Mechanism: by reference

Is the DATATRIEVE Access Block.

Usage Notes

- The call DTR\$PRINT_DAB enables you to display the contents of the DAB fields at any time during execution of your program. You can use DTR\$PRINT_DAB to determine the stallpoint, condition, and message or print line that result from each call.
- When you call DTR\$PRINT_DAB, DATATRIEVE displays up to 80 characters of the message buffer (DAB\$A_MSG_BUF) and up to 80 characters of the auxiliary message buffer (DAB\$A_AUX_BUF), regardless of the lengths of DAB\$W_MSG_LEN and DAB\$W_AUX_LEN. To print the entire contents of either buffer when it holds a message longer than 80 characters, use program code to display the text. For example, the following BASIC code displays the contents of the DAB and the entire contents of the modified message and auxiliary buffers:

DTR\$PRINT_DAB

```
! Declare new message and auxiliary buffers.
MAP (NEW_BUFFERS) STRING NEW_MSG_BUFF = 132,      &
                          NEW_AUX_BUFF = 132

RETURN_STATUS = DTR$INIT (DAB, 100%, NEW_MSG_BUFF, NEW_AUX_BUFF,)
                    .
                    .
                    .

! Display the contents of the DAB.
RETURN_STATUS = DTR$PRINT_DAB (DAB)

! Display the full message and auxiliary buffers.
PRINT "Message buffer: "; NEW_MSG_BUFF
PRINT "Auxiliary buffer: "; NEW_AUX_BUFF
```

Return Status

SS\$NORMAL

Call completed successfully.

DTR\$BADNUMARG

The DAB is invalid.

Other errors from RMS, system services, and Run-Time Library routines.

Example

Determine the error in the following sequence of FORTRAN statements:

```
1      CALL DTR$COMMAND (DAB, 'READY PERSONNEL;')
2      CALL DTR$COMMAND (DAB, 'PRINT PERSONNEL;')
3      CALL DTR$FINISH (DAB)
```

These statements do not produce the desired result: DATATRIEVE does not print any PERSONNEL records.

Insert the call DTR\$PRINT_DAB between statements 1 and 2 and run your program. DATATRIEVE displays the following information:

DTR\$PRINT_DAB

VAX DATATRIEVE Access Block Dump

```
DAB Address : %X'000004B0'  
DAB$L_CONDITION : 9274723 (%X'008D8563') DTR$_SUCCESS (I)  
DAB$A_MSG_BUF : %X'00000400' DAB$W_MSG_BUF_LEN : 80 DAB$W_MSG_LEN : 34  
Statement completed successfully.  
DAB$A_AUX_BUF : %X'00000450' DAB$W_AUX_BUF_LEN : 20 DAB$W_AUX_LEN : 0  
  
DAB$W_IDI : 0 DAB$W_STATE : 4 (DTR$K_STL_MSG)  
DAB$L_FLAGS : %X'00000000'  
  
DAB$L_OPTIONS : %X'00000091'  
DAB$V_SEMI_COLON_OPT DAB$V_UNQUOTED_LIT DAB$V_FORMS_ENABLE  
  
DAB$W_REC_LEN : 0 DAB$W_VERSION.DAB$W_LEVEL : 4.1  
DAB$W_UDK_INDEX : 0 DAB$W_COLUMNS_PAGE : 80 DAB$W_TT_CHANNEL : 224  
DAB$W_CTLC_CHANNEL : 0 DAB$L_KEYTABLE_ID : 1866504  
DAB$L_COMMAND_KEYBOARD : 1893432 DAB$L_PROMPT_KEYBOARD : 1894184
```

The DTR\$_SUCCESS message shows that DATATRIEVE has executed the READY command in the first call. The value of DAB\$W_STATE shows that DATATRIEVE is currently at the message stallpoint (DTR\$K_STL_MSG), that is, it has a message.

DATATRIEVE does not execute the PRINT statement in the second call until your program handles the message stallpoint. The simplest way to do this is to call DTR\$DTR or DTR\$WINDOWS with the DTR\$M_OPT_CMD option. DATATRIEVE displays its message (if it is anything other than DTR\$_SUCCESS) and returns control to your program when it is at the command stallpoint (DTR\$K_STL_CMD).

After you insert the call DTR\$DTR or DTR\$WINDOWS between the first and second calls, DATATRIEVE still does not execute the PRINT statement. If you use the call DTR\$PRINT_DAB after the second call, the resulting DAB dump shows that you are at the print line stallpoint (DTR\$K_STL_LINE). DATATRIEVE has some print lines, and your program must handle them. Again, the simplest way to have DATATRIEVE execute your call is to call DTR\$DTR or DTR\$WINDOWS.

DTR\$PUT_OUTPUT

DTR\$PUT_OUTPUT

Writes a line to a file created by the DATATRIEVE OPEN command.

Format

DTR\$PUT_OUTPUT (dab, string, [prompt-string])

Arguments

dab

Data type: data block

Access: read/write

Mechanism: by reference

Is the DATATRIEVE Access Block.

string

Data type: character string

Access: read-only

Mechanism: by descriptor

Is the string you want to write to the output file.

prompt-string

Data type: character string

Access: read-only

Mechanism: by descriptor

Is a string inserted before the line specified by the string argument.

Usage Notes

- If you create a log file with the DATATRIEVE OPEN statement, DATATRIEVE writes the string you specify to that file.
- You can use this call at any stallpoint.

DTR\$PUT_OUTPUT

Return Status

SS\$ _NORMAL

Call completed successfully.

DTR\$ _BADNUMARG

Invalid number of arguments.

Other errors from RMS, system services, and Run-Time Library routines.

Example

You are using the BASIC program UDK (listed in the examples of the DTR\$CREATE_UDK call). You would like to use the DATATRIEVE OPEN command to write the results of the SHOW UDKS command to a log file. Use the call DTR\$PUT_OUTPUT as follows:

```
      .
      .
      RET_STATUS = DTR$CREATE_UDK (DAB BY REF, 'UDKS', 5% BY REF, &
      DTR$K_UDK_SHOW BY REF)

      ! Declare the options for the DTR$DTR call.
2000  DTR_OPTIONS = DTR$M_OPT_UDK ! Return to program on a UDK &
      + DTR$M_OPT_CONTROL_C &
      + DTR$M_OPT_STARTUP &
      + DTR$M_OPT_FOREIGN &
      + DTR$M_OPT_BANNER

      ! Call the terminal server.
2500  RET_STATUS = DTR$DTR ( DAB BY REF, DTR_OPTIONS BY REF )
      ! Check for EXIT or CTRL/Z.
      GOTO 6000 IF RET_STATUS = DTR$_EXIT

      ! UDK 5 - User entered SHOW UDKS.
3500  PRINT " "
      PRINT " User-Defined Keywords Available "
      PRINT " "
      PRINT " CLEAR_SCREEN - clears the screen "
      PRINT " DIRECTORY - displays files in the default directory"
      PRINT " SPAWN - creates a subprocess "
      PRINT " MAIL - invokes VMS MAIL "

      ! If user entered OPEN file-spec, write the preceding text
      ! out to the log file.
```

DTR\$PUT_OUTPUT

```
RET_STATUS = DTR$PUT_OUTPUT (DAB BY REF, " " BY DESC)
RET_STATUS = DTR$PUT_OUTPUT (DAB BY REF, " User-Defined "+ &
    "Keywords Available " BY DESC)
RET_STATUS = DTR$PUT_OUTPUT (DAB BY REF, " " BY DESC)
RET_STATUS = DTR$PUT_OUTPUT (DAB BY REF, " CLEAR_SCREEN - "+ &
    "clears the screen " BY DESC)
RET_STATUS = DTR$PUT_OUTPUT (DAB BY REF, " SPAWN - "+ &
    "creates a subprocess " BY DESC)
RET_STATUS = DTR$PUT_OUTPUT (DAB BY REF, " MAIL - "+ &
    "invokes VMS MAIL " BY DESC)

! End the UDK.
RET_STATUS = DTR$END_UDK BY REF (DAB)

GOTO 2500
6000 END
```

DTR\$PUT_PORT

DTR\$PUT_PORT

Passes a record from your program to DATATRIEVE.

Format

DTR\$PUT_PORT (dab, record-buffer)

Arguments

dab

Data type: data block

Access: read/write

Mechanism: by reference

Is the DATATRIEVE Access Block.

record-buffer

Data type: data block

Access: read-only

Mechanism: by reference

Is a buffer that contains the record being passed to DATATRIEVE.

Usage Notes

- To transfer records from your program to DATATRIEVE, you must define a record buffer in your program.
- If you write your programs in a high-level language that supports the dictionary (such as BASIC, COBOL, and FORTRAN), you need not code the entire record buffer into your program. You can copy a record definition from the dictionary and use it as a record buffer.
Be sure that your record definition does not contain a field name that is a reserved word in the programming language you are using.
- To transfer records to DATATRIEVE, you must also define a port. A port is a single record buffer that can be referenced by DATATRIEVE and your program. All transfer of records between DATATRIEVE and the host program is done through ports. To define a port, use the DEFINE PORT command or the DECLARE PORT statement.

DTR\$PUT_PORT

- The **DEFINE PORT** command inserts your port definition into the dictionary. In the **DEFINE PORT** command, you specify a name for the port and an associated record definition in the following format:

```
DEFINE PORT path-name [USING] record-path-name;
```

To define a port for **YACHTS** records, you can use the following command in interactive **DATATRIEVE**:

```
DTR> DEFINE PORT BOAT_PORT USING YACHT;
```

You can also use the **DTR\$COMMAND** call to pass this command to **DATATRIEVE** from your program. If you use **DEFINE PORT** to create a port, you must ready the port before using it.

- The **DECLARE PORT** statement creates a temporary port with the name you specify and readies the port for write access. **DATATRIEVE** does not enter a definition of the port in the dictionary. You can issue the **DECLARE PORT** statement only from your program. You must define a record in the statement.

To declare a port for **YACHTS** records in a **FORTRAN** program, you can use the following statement:

```
CALL DTR$COMMAND (DAB,'DECLARE PORT BOAT_PORT USING
1 01 YACHT.
2    03 BOAT.
3      06 BUILDER PIC X(10).
4      06 MODEL PIC X(10).
5      06 RIG PIC X(6).
6      06 LOA PIC X(3).
7      06 DISP PIC X(5).
8      06 BEAM PIC XX.
9      06 PRICE PIC X(5).;')
```

- If your program reads in data from the terminal, you may need to declare all fields of your port as character string data types.
- To pass records to **DATATRIEVE**, use a **DATATRIEVE** statement that specifies a port; for example:

```
FOR BOAT_PORT STORE YACHTS USING BOAT = BOAT
FOR PT1 MODIFY YACHTS USING TYPE = PT1_TYPE
```

- When you tell **DATATRIEVE** to store records from a port into a domain, **DATATRIEVE** enters the put port stallpoint (**DTR\$K_STL_PPUT**) and waits for the program to pass a record. You can pass a record to **DATATRIEVE** with the call **DTR\$PUT_PORT**. You can terminate the passing of records with the call **DTR\$PORT_EOF**.

DTR\$PUT_PORT

- After each DTR\$PUT_PORT call, DATATRIEVE is usually at the put port or message stallpoint (DTR\$K_STL_PPUT or DTR\$K_STL_MSG). If a DTR\$PUT_PORT call causes an error, DATATRIEVE enters the message stallpoint (DTR\$K_STL_MSG) and informs your program of the error.
- The message buffer contains the name of the port when DATATRIEVE is at the put port stallpoint (DTR\$K_STL_PPUT).
- You cannot sort or reduce records passed to DATATRIEVE from a port.

Return Status

SS\$_NORMAL

Call completed successfully.

DTR\$_BADHANDLE

The DAB is invalid.

DTR\$_WRONGSTALL

Wrong stallpoint for this call. You must be at the DTR\$K_STL_PPUT stallpoint.

Other errors from RMS, system services, and Run-Time Library routines.

Example

For an example of how to use DTR\$PUT_PORT, see the program PUTPORT in the examples to the DTR\$PORT_EOF call.

DTR\$PUT_VALUE

DTR\$PUT_VALUE

Passes a value to DATATRIEVE.

Format

DTR\$PUT_VALUE (dab, [value])

Arguments

dab

Data type: data block

Access: read/write

Mechanism: by reference

Is the DATATRIEVE Access Block.

value

Data type: character string

Access: read-only

Mechanism: by descriptor

Is a string you want to pass to DATATRIEVE.

Usage Notes

- If you pass DATATRIEVE a statement that stores or modifies fields or a statement that contains a prompting expression, DATATRIEVE enters the prompt stallpoint (DTR\$K_STL_PRMP) and returns control to your program. Your program must provide a value for the prompting expression before DATATRIEVE can continue. To pass a value back to DATATRIEVE, use the call DTR\$PUT_VALUE.
- The value passed must be an ASCII string.
- The DATATRIEVE prompt is stored in the message buffer.
- If you are modifying or storing a field, the name of the field is stored in the auxiliary message buffer.
- If you use a prompting expression, that expression is stored in the auxiliary message buffer. For example, suppose you pass DATATRIEVE the statement:

```
FIND PERSONNEL WITH LAST_NAME CONT *."last name"
```


DTR\$PUT_VALUE

The string “last name” is stored in the auxiliary message buffer. The string “Enter last name” is stored in the message buffer.

- If you do not pass the value argument, the value of the field you are storing or modifying is not changed. Not passing the value argument is equivalent to pressing the TAB key in response to a prompt in interactive DATATRIEVE.

Return Status

SS\$_NORMAL

Call completed successfully.

DTR\$_BADHANDLE

The DAB is invalid.

DTR\$_BADNUMARG

Invalid number of arguments.

DTR\$_BADSTRDES

Invalid string descriptor.

Other errors from RMS, system services, and Run-Time Library routines.

Example

Read a value and pass it to DATATRIEVE from a FORTRAN program:

```
CHARACTER*20 VALUE

      DTR_OPTIONS = ! Set DTR$DTR options.
      1          + DTR$_M_OPT_CMD
      2          + DTR$_M_OPT_PRMP

C Display the DATATRIEVE value prompt and read a value.
      DO WHILE (DAB$_W_STATE .EQ. DTR$_K_STL_PRMP)
100      WRITE (6, 150) MSG_BUFF
150      FORMAT (1X, A<DAB$_W_MSG_LEN>, $)
      READ (5, 1) VALUE
1      FORMAT (A)

C Pass the value entered to DATATRIEVE.
      RET_STATUS = DTR$PUT_VALUE (DAB, VALUE)

C Use DTR$DTR to display messages. Return control to the program
C if there are more values to be entered or if DATATRIEVE is
C ready for the next command.
```

DTR\$PUT_VALUE

```
RET_STATUS = DTR$DTR (DAB, DTR_OPTIONS)  
END DO
```

DTR\$UNWIND

DTR\$UNWIND

Stops execution of a command or statement passed to DATATRIEVE.

Format

DTR\$UNWIND (dab)

Argument

dab

Data type: data block

Access: read/write

Mechanism: by reference

Is the DATATRIEVE Access Block.

Usage Notes

- This call is useful if your program allows users to interact with DATATRIEVE. Your program can check stallpoint information to decide whether to abort a command string.
- DTR\$UNWIND has the same effect as CTRL/Z during execution of a STORE statement or CTRL/C during execution of a PRINT statement in interactive DATATRIEVE.
- When you use the call DTR\$UNWIND, DATATRIEVE does not immediately terminate the statement you are unwinding. DATATRIEVE sets a flag in the DAB, which records your DTR\$UNWIND call. When you return control to DATATRIEVE, DATATRIEVE sets DAB\$L_CONDITION to DTR\$_UNWIND, terminates the command, and stalls at the message stallpoint (DTR\$K_STL_MSG). The message buffer contains the message: "Execution terminated by operator."

Return Status

SS\$ NORMAL

Call completed successfully.

DTR\$ BADHANDLE

The DAB is invalid.

Other errors from RMS, system services, and Run-Time Library routines.

DTR\$UNWIND

Example

Use DTR\$UNWIND to terminate a STORE statement in a FORTRAN program:

```
INCLUDE 'DTR$LIBRARY:DAB'
CHARACTER*20 VALUE, ANS

RET_STATUS = DTR$INIT (DAB, 100, MSG_BUFF, AUX_BUFF,
1      DTR$K_SEMI_COLON_OPT)

RET_STATUS = DTR$COMMAND (DAB, 'READY YACHTS WRITE')

C Use the DTR$DTR call to handle the "Statement completed
C successfully" message and to display error messages.

RET_STATUS = DTR$DTR (DAB, DTR$M_OPT_CMD)

100  RET_STATUS = DTR$COMMAND (DAB, 'REPEAT 5 STORE YACHTS')

DTR_OPTIONS =          ! Select DTR$DTR options
1      + DTR$M_OPT_CMD    ! Return on DTR$K_STL_CMD
2      + DTR$M_OPT_PRMP  ! Return on DTR$K_STL_PRMP

DO WHILE (DAB$W_STATE .EQ. DTR$K_STL_PRMP)

C Display the DATATRIEVE "Enter field-name" prompt.

120      WRITE (6, 150) MSG_BUFF
150      FORMAT (1X, A<DAB$W_MSG_LEN>, $)

C Read the value entered. At end of STORE, or if user
C aborts the STORE statement with CTRL/Z, go to 999.

      READ (5, 160, END = 999) VALUE
160      FORMAT (A)

C Pass the value entered to DATATRIEVE.

      RET_STATUS = DTR$PUT_VALUE (DAB, VALUE)

C If a validation error occurs, display message and reprompt.

      IF(DAB$W_STATE .EQ. DTR$K_STL_MSG) THEN
          RET_STATUS = DTR$DTR (DAB, DTR_OPTIONS)
      END IF
END DO

C If user entered CTRL/Z to a prompt, use the DTR$UNWIND call.
C DATATRIEVE records this call but does not execute it.

999  IF (DAB$W_STATE .NE. DTR$K_STL_CMD) THEN
      RET_STATUS = DTR$UNWIND (DAB)

C Use the DTR$PUT_VALUE call again. DATATRIEVE executes the
C DTR$UNWIND call.

      RET_STATUS = DTR$PUT_VALUE (DAB, VALUE)
```

DTR\$UNWIND

C Use DTR\$DTR to display messages.

```
      RET_STATUS = DTR$DTR (DAB, DTR$M_OPT_CMD)
END IF
WRITE (6, *) 'Do you wish to continue storing? Y or N:'
READ (5, 160) ANS
IF ((ANS(1:1) .EQ. 'Y') .OR. (ANS(1:1) .EQ. 'y')) THEN
      GO TO 100
END IF
RET_STATUS = DTR$FINISH (DAB)
END
```

DTR\$WINDOW_MSG

DTR\$WINDOW_MSG

Displays a user-created message in a DATATRIEVE DECwindows interface message box. This call takes advantage of the DECwindows message dialog box format to display a message from your program. The DECwindows message dialog box requires that the user acknowledge the message by clicking on the ACKNOWLEDGE button in the message box.

Format

DTR\$WINDOW_MSG (dab, message-string)

Argument

dab

Data type: data block
Access: read/write
Mechanism: by reference

Is the DATATRIEVE Access Block.

message-string

Data type: character string
Access: read-only
Mechanism: by descriptor

Usage Note

You must have made a call to DTR\$WINDOWS to activate the DATATRIEVE DECwindows terminal server before you can call DTR\$WINDOW_MSG.

Return Status

SS\$NORMAL

Call completed successfully.

DTR\$BADHANDLE

The DAB is invalid.

DTR\$BADNUMARG

Invalid number of arguments.

DTR\$WINDOW_MSG

DTR\$_BADSTRDES

Invalid string descriptor.

DTR\$_WRONGSTALL

Wrong stallpoint for this call. The stallpoint must be DTR\$K_STL_CMD.

DTR\$_NOWINDOWS

The DATATRIEVE DECwindows terminal server has not been activated by a call to DTR\$WINDOWS.

Other errors from RMS, system services, and Run-Time Library routines.

DTR\$WINDOW_OUTPUT

DTR\$WINDOW_OUTPUT

Lets the user display a line of up to 255 characters in the DATATRIEVE main application window. You can use this call in your program to print out a line of text in the output area of the DATATRIEVE main application window.

Format

DTR\$WINDOW_OUTPUT (dab, text-string)

Arguments

dab

Data type: data block
Access: read/write
Mechanism: by reference

Is the DATATRIEVE Access Block.

text-string

Data type: character string
Access: read-only
Mechanism: by descriptor

Is a text string of up to 255 printable characters or spaces. The text string should not include ASCII characters with a decimal value of less than 32, or control characters such as line feeds or tabs.

Usage Note

Your program must have made a call to DTR\$WINDOWS to activate the DATATRIEVE DECwindows terminal server before you can use DTR\$WINDOW_OUTPUT.

Return Status

SS\$ _NORMAL

Call completed successfully.

DTR\$ _BADHANDLE

The DAB is invalid.

DTR\$WINDOW_OUTPUT

DTR\$_BADNUMARG

Invalid number of arguments.

DTR\$_BADSTRDES

Invalid string descriptor or string is longer than 255 characters.

DTR\$_WRONGSTALL

Wrong stallpoint for this call. The stallpoint must be DTR\$K_STL_CMD.

DTR\$_NOWINDOWS

The DATATRIEVE DECwindows terminal server has not been activated by a call to DTR\$WINDOWS.

Other errors from RMS, system services, and Run-Time Library routines.

DTR\$WINDOWS

DTR\$WINDOWS

Invokes the DATATRIEVE DECwindows terminal server. Your program gets access to all of the DATATRIEVE interactive data management capabilities in a DECwindows environment. Users of your program cannot tell that they are running a program and not interactive DATATRIEVE. The DTR\$WINDOWS call performs the same function as the DTR\$DTR call, but should be used in place of DTR\$DTR to use the DECwindows terminal server.

Format

DTR\$WINDOWS (dab, options-code)

Arguments

dab

Data type: data block
Access: read/write
Mechanism: by reference
Is the DATATRIEVE Access Block.

options-code

Data type: longword integer
Access: read-only
Mechanism: by reference

Is a bit mask of options you select, where each bit in the options-code argument identifies a separate option. The default for options-code is zero.

The following tables describe the DTR\$WINDOWS options and list their values. Table 3–6 describes the DTR\$WINDOWS options that determine when the DATATRIEVE terminal server returns control to your program. Table 3–7 describes the DTR\$WINDOWS options that enable various DATATRIEVE terminal server functions.

Table 3–6 DTR\$WINDOWS Control Options

Option	Value	DECwindows terminal server returns control to your program when:
DTR\$M_OPT_CMD	1	Stallpoint is DTR\$K_STL_CMD

(continued on next page)

DTR\$WINDOWS

Table 3–6 (Cont.) DTR\$WINDOWS Control Options

Option	Value	DECwindows terminal server returns control to your program when:
DTR\$M_OPT_PRMPPT	2	Stallpoint is DTR\$K_STL_PRMPPT
DTR\$M_OPT_LINE	4	Stallpoint is DTR\$K_STL_LINE
DTR\$M_OPT_MSG	8	Stallpoint is DTR\$K_STL_MSG
DTR\$M_OPT_PGET	16	Stallpoint is DTR\$K_STL_PGET
DTR\$M_OPT_PPUT	32	Stallpoint is DTR\$K_STL_PPUT
DTR\$M_OPT_CONT	64	Stallpoint is DTR\$K_STL_CONT
DTR\$M_OPT_UDK	128	Stallpoint is DTR\$K_STL_UDK
DTR\$M_OPT_DTR_UDK	256	User enters a DATATRIEVE keyword
DTR\$M_OPT_END_UDK	512	Stallpoint is DTR\$K_STL_END_UDK
DTR\$M_OPT_UNWIND	1024	Condition is DTR\$_UNWIND—user enters CTRL/C or CTRL/Z during execution of a command or statement

Table 3–7 DTR\$WINDOWS DECwindows Terminal Server Options

Option	Value	This option enables:
DTR\$M_OPT_CONTROL_C	2048	DATATRIEVE CTRL/C handling
DTR\$M_OPT_STARTUP	4096	Execution of the startup command file the first time you call DTR\$WINDOWS with this option
DTR\$M_OPT_FOREIGN	8192	Execution of the foreign command line

Usage Notes

- If you have made a call to DTR\$WINDOWS, you cannot make a call to DTR\$DTR until you have called DTR\$FINISH_WINDOWS. DTR\$FINISH_WINDOWS terminates your session with the DATATRIEVE DECwindows terminal server. DTR\$DTR invokes the DATATRIEVE terminal server.

DTR\$WINDOWS

- If you have made a call to DTR\$DTR, you cannot make a call to DTR\$WINDOWS until you have called DTR\$FINISH. DTR\$FINISH terminates your session with the DATATRIEVE terminal server. DTR\$WINDOWS invokes the DATATRIEVE DECwindows terminal server.
- To end your program's interaction with the DATATRIEVE DECwindows terminal server, use the DTR\$FINISH_WINDOWS call. The DTR\$FINISH_WINDOWS call cannot be used unless the DATATRIEVE DECwindows terminal server has been activated by a call to DTR\$WINDOWS.
- The calls that enable and disable CTRL/C operation (DTR\$DISABLE_CONTROL_C and DTR\$ENABLE_CONTROL_C) are invalid if you have invoked the DATATRIEVE DECwindows terminal server with the DTR\$WINDOWS call.
- The DATATRIEVE DECwindows terminal server does not allow you to use the SMG (Screen Management Facility) interface or the virtual keyboards for command line and prompting expressions.
- You can use the options-code argument to specify the stallpoints you want the terminal server to handle and the stallpoints at which you want your program to take control. For example, you can call DTR\$WINDOWS to have DATATRIEVE display messages and print lines and then return control to your program by specifying DTR\$M_OPT_CMD in the options-code argument.
- If your program gives control to the DATATRIEVE DECwindows terminal server and the user ends the DATATRIEVE session by entering CTRL/Z or EXIT in response to the DTR> prompt or by selecting the Exit menu item of the File menu, control returns to your program.
- If you specify the option DTR\$M_OPT_UNWIND, DATATRIEVE returns control to your program whenever it detects an unwind condition. An unwind condition occurs when the user enters CTRL/Z, or when DATATRIEVE encounters a DTR\$UNWIND call in your program.
- DTR\$WINDOWS assigns a channel number to the DAB\$W_TT_CHANNEL DAB field if a number has not been previously assigned. DATATRIEVE uses the value of DAB\$W_TT_CHANNEL as the input/output channel for TDMS forms and Guide Mode. To operate with forms (namely FMS, TDMS, and DECforms), it is required a value different from zero. (For information about deassigning channels, see Chapter 2.)
- You can use DTR\$WINDOWS with the call DTR\$CREATE_UDK to add your own keywords to interactive DATATRIEVE (see the discussion of DTR\$CREATE_UDK in this chapter).

DTR\$WINDOWS

- You can specify the option `DTR$M_OPT_STARTUP` to invoke a startup command file pointed to by the logical `DTR$STARTUP`. This command file is executed only the first time you call `DTR$WINDOWS` with the `DTR$M_OPT_STARTUP` option set.

Return Status

SS\$_NORMAL

Call completed successfully.

DTR\$_BADNUMARG

Invalid number of arguments.

DTR\$_BADSTALL

Invalid stallpoint in the DAB.

DTR\$_EXIT

User entered either `EXIT` or `CTRL/Z`.

DTR\$_INVOPTION

User has specified one of the following invalid options:

```
DTR$M_OPT_BANNER
DTR$M_OPT_KEYDEFS
DTR$M_OPT_REMOVE_CTLC
```

DTR\$_NOWINDOWS

User tried to invoke the `DATATRIEVE` DECwindows terminal server with a `DTR$WINDOWS` call while the `DATATRIEVE` terminal server (invoked by the `DTR$DTR` call) is still active.

DTR\$_UNWIND

Program called `DTR$UNWIND` or user entered `CTRL/C` or `CTRL/Z` to terminate `DATATRIEVE` statement.

Other errors from RMS, system services, and Run-Time Library routines.

Examples

Use `DTR$WINDOWS` to handle error messages in a FORTRAN program:

```
IF (DAB$L_CONDITION .NE. %LOC(DTR$_SUCCESS))
1     STATUS = DTR$WINDOWS (DAB, DTR$M_OPT_CMD)
```

DTR\$WINDOWS

Use DTR\$WINDOWS to display print lines in a COBOL program:

```
IF DTR$K_STL_LINE THEN
    STATUS = "DTR$WINDOWS" USING DAB DTR$M_OPT_CMD GIVING STATUS.
```

Use DTR\$WINDOWS in a BASIC program to simulate interactive DATATRIEVE. Note that the option DTR\$M_OPT_STARTUP executes a DATATRIEVE startup command file, if you have one.

```
100    %INCLUDE "DTR$LIBRARY:DAB"

        ! Declare the initialization and terminal server calls as
        ! functions.

EXTERNAL INTEGER FUNCTION DTR$INIT, DTR$WINDOWS
DECLARE INTEGER DTR_OPTIONS, RET_STATUS

        ! Declare the normal and exit status.

EXTERNAL INTEGER CONSTANT DTR$_BADNUMARG, &
                        DTR$_BADSTALL, &
                        DTR$_EXIT, &
                        SS$_NORMAL

        ! Initialize the interface.
500    RET_STATUS = DTR$INIT (DAB BY REF, 100% BY REF, MSG_BUFF, &
                        AUX_BUFF, DTR$K_SEMI_COLON_OPT BY REF )

        ! Check to see if DATATRIEVE was initialized.
IF RET_STATUS <> SS$_NORMAL THEN
    PRINT "DATATRIEVE initialization failed."
    GOTO 2000

        ! Set options include commands to: execute a startup file,
        ! enable CTRL/C handling, allow invocation command lines,
        ! and display a startup banner.
DTR_OPTIONS    =    DTR$M_OPT_STARTUP    &
                OR DTR$M_OPT_FOREIGN

        ! Call the terminal server.
1000    RET_STATUS = DTR$WINDOWS (DAB BY REF, DTR_OPTIONS BY REF)

        ! Check the status.

SELECT RET_STATUS
CASE SS$_NORMAL
    PRINT "Check your DTR$WINDOWS options."
    PRINT "You returned control to the program."
CASE DTR$_EXIT
    PRINT "Bye."
CASE ELSE
    CALL LIB$SIGNAL (RET_STATUS BY VALUE)
END SELECT
```

DTR\$WINDOWS

```
1500  RET_STATUS = DTR$FINISH_WINDOWS BY REF (DAB)
2000  END
```

You can use this program as the framework for customizing interactive DATATRIEVE.

4

Adding Functions to DATATRIEVE

A DATATRIEVE **function** is a mechanism to produce values you define and add to the DATATRIEVE language. By adding functions, you extend the capability of DATATRIEVE to perform specific tasks efficiently.

This chapter explains how to add functions to DATATRIEVE. In order to add functions, you must install a new shareable image. This requires the VMS privileges SYSPRV and CMKRNL.

4.1 Using DATATRIEVE Functions

You can use a function within DATATRIEVE as a value expression in a statement. For example, to calculate and print the square root of a number, you can use the function FN\$SQRT in a PRINT statement:

```
DTR> FIND FIRST 5 YACHTS; SELECT 1
DTR> PRINT LOA, FN$SQRT(LOA) ("SQUARE ROOT"/"OF LOA")

LENGTH
OVER  SQUARE ROOT
ALL   OF LOA
37    6.08276271
```

You can also use functions to assign values to variables and in virtual field definitions. For example, to calculate the hexadecimal equivalent of a decimal number, you can use the function FN\$HEX within a COMPUTED BY clause:

```
DTR> DECLARE A COMPUTED BY FN$HEX(B) .
DTR> DECLARE B PIC 99.
DTR> B = 16
DTR> PRINT A
```

A

Adding Functions to DATATRIEVE

4.1 Using DATATRIEVE Functions

You can nest functions, as the following example shows:

```
DTR> PRINT FN$HEX(FN$SQRT(2500))
      FN$HEX
      32
```

The examples in this section show how to use functions to form value expressions. Functions can also serve as DATATRIEVE commands and statements. For example, the function FN\$CREATE_LOG defines a process logical name that is valid only during the current DATATRIEVE session. FN\$CREATE_LOG accepts two arguments: a logical name and an equivalence name.

The following example shows how to use a function to create a logical name within DATATRIEVE:

```
DTR> FN$CREATE_LOG('FILE', -
CON> 'BRANDY"GREBE NANCY":DB2:[GREBE]REPORT.LIS')
DTR> REPORT ON FILE
      .
      .
      .
RW> END_REPORT
DTR>
```

4.2 DATATRIEVE Functions and External Procedures

A DATATRIEVE function invokes an external procedure. The procedure is a unit of code designed to perform a specific task. The procedure that a DATATRIEVE function invokes can be one of two types:

- A Run-Time Library procedure
- A procedure that you write

In the first example in the preceding section, the DATATRIEVE function FN\$SQRT calls the Run-Time Library procedure MTH\$SQRT. MTH\$SQRT performs the square root calculation and returns the value to FN\$SQRT.

FN\$SQRT, FN\$HEX, and FN\$CREATE_LOG are some of the functions included in the DATATRIEVE installation kit. The *VAX DATATRIEVE Reference Manual* describes these sample functions. The rest of this chapter explains how you can use Run-Time Library procedures and procedures you write to add your own functions to DATATRIEVE.

Adding Functions to DATATRIEVE

4.3 How to Add Functions to DATATRIEVE

4.3 How to Add Functions to DATATRIEVE

To add functions to DATATRIEVE, you must use several of the following files, which are located in DTR\$LIBRARY.

DTRFNDxx.MAR ¹	The VAX MACRO file that contains user-defined function definitions
DTRFUNxx.OLB	The library file that contains the object module DTRFND and object modules of user-written procedures
DTRBLDxx.COM	The command file that links the DATATRIEVE shareable image

¹xx represents the 1 to 26-character suffix that may be added at installation. DTRBLDxx might be DTRBLDFMS, for example.

The following steps are necessary to add functions to DATATRIEVE. If you are writing your own procedure, begin at step 1. If you are using a Run-Time Library procedure, begin at step 3.

1. Write your procedures and compile them.
2. Insert the procedure object files into DTRFUNxx.OLB.
3. Add the function definitions to DTRFNDxx.MAR.
4. Assemble DTRFNDxx.MAR, creating DTRFNDxx.OBJ.
5. Replace DTRFNDxx.OBJ in DTRFUNxx.OLB.
6. Relink the DATATRIEVE shareable image.

The functions supplied as part of the DATATRIEVE kit are stored in the the VAX MACRO file IDTRFND.MAR. If you want to replace one of the DATATRIEVE supplied functions with one that you create, you should follow the steps listed previously in this section. The new function should have the same name as the DATATRIEVE function you wish to replace. When you relink the DATATRIEVE shareable image, DATATRIEVE checks against the contents of IDTRFND.MAR to look for a match. If it finds a match, DATATRIEVE overrides the function defined in IDTRFND.MAR and, instead, refers to the new function defined in DTRFNDxx.MAR.

Note

Do not modify the contents on IDTRFND.MAR. The code for defining functions that are supplied with the DATATRIEVE kit is different from the code used to define user-defined functions.

Adding Functions to DATATRIEVE

4.3 How to Add Functions to DATATRIEVE

For additional information on using VMS RTLs to create and add DATATRIEVE functions, refer to the VMS documentation set. It contains a master index with references to information on the Run-Time Library, modular library procedures, systems services, and utilities.

In addition, the *VAX DATATRIEVE Installation Guide* contains information on linking and installing DATATRIEVE.

You may also want to refer to the reference manual of the language in which you write your own procedures.

4.3.1 Write and Compile Your Procedures

Before you write a procedure, refer to the VMS documentation set for procedures included in the Run-Time Library. If you find the procedure you want in the Run-Time Library, you can go to Section 4.3.3, Add the Function Definitions to DTRFNDxx.MAR, later in this chapter.

If you decide to write your own procedure, you must observe the following restrictions:

- All procedures should be read only.
If a procedure is not read only and more than one DATATRIEVE user uses the function that calls it, then invalid results can occur.
- All procedures must contain position-independent code.
For DTRSHR.EXE to be position-independent, each function linked with it must also be position-independent. If you write procedures in C, BASIC, Pascal, or PL/I, the resulting code is always position-independent. BLISS and MACRO produce position-independent code if you use self-relative addressing.
FORTRAN and COBOL do not produce position-independent code. However, the VMS Linker makes FORTRAN and COBOL code position-independent. Therefore, you can write procedures in FORTRAN and COBOL and link them with DTRSHR.

The following procedure is written in BASIC. Its purpose is to pass to DATATRIEVE the value of a number raised to a power.

```
10      FUNCTION REAL POWER ( REAL X, Y)
20      POWER = X ** Y
30      FUNCTIONEND
```

The following procedure is written in FORTRAN. It enables you to pass an escape sequence as a text string to DATATRIEVE. This particular escape sequence clears the screen on a VT100 terminal.

Adding Functions to DATATRIEVE

4.3 How to Add Functions to DATATRIEVE

```
INTEGER FUNCTION CLEAR (SCREEN)
CHARACTER*1 ESC
CHARACTER*7 SCREEN
ESC      = CHAR(27)
SCREEN  = ESC//'[2J'//ESC//[H'
CLEAR   = 1
END
```

The VMS Linker searches the module library DTR\$LIBRARY:DTRFUNxx.OLB to resolve references to external procedure names such as POWER and CLEAR. If you want to call these procedures from within DATATRIEVE, the first step is to compile them:

```
$ BASIC POWER
$ FORTRAN CLEAR
```

4.3.2 Insert the Procedure Object Files into the Object Library

After you compile your procedures, use the LIBRARY command to insert the object modules into DTRFUNxx.OLB:

```
$ LIBRARY/INSERT DTR$LIBRARY:DTRFUNxx POWER, CLEAR
```

4.3.3 Add the Function Definitions to DTRFNDxx.MAR

User-defined functions are contained in the file DTRFNDxx.MAR. DTRFNDxx.MAR must always contain at least one function. (The function FN\$GET_SYMBOL is included in DTRFNDxx.MAR as part of the DATATRIEVE kit. Do not remove FN\$GET_SYMBOL from DTRFNDxx.MAR unless you have replaced it with another function definition.) DTRFNDxx.MAR is coded in VAX MACRO. It uses macros that are defined in DTR\$LIBRARY:DTRFNLB.MLB. You do not need to understand VAX MACRO to add function definitions.

In order to define your own functions, edit DTRFNDxx.MAR and insert the new function definitions. The following sections explain the format and the parts of a function definition and provide several sample function definitions.

4.3.3.1 The Format of a Function Definition

A function definition has the following format:

Adding Functions to DATATRIEVE

4.3 How to Add Functions to DATATRIEVE

```

$DTR$FUN_DEF  internal-name, external-name, number-of-arguments
[$DTR$FUN_HEADER  HDR=<"header-segment"[/...]>]
[$DTR$FUN_EDIT_STRING  ^\edit-string\]

$DTR$FUN_OUT_ARG TYPE = {
    FUN$K_STATUS
    FUN$K_INPUT
    FUN$K_VALUE ,DTYPE = data-type
}

$DTR$FUN_IN_ARG TYPE = {
    FUN$K_NULL
    FUN$K_TEXT ,OUT_PUT = TRUE [,ALL_LEN = m]
    FUN$K_VALUE ,DTYPE = data-type ,ORDER = p
    FUN$K_DESC ,DTYPE = data-type ,ORDER = p[, OUT_PUT = TRUE],
    FUN$K_REF ,DTYPE = data-type {
        , OUT_PUT = TRUE[, ORDER = p]
        , ORDER = p
    }
} [...]

[$DTR$FUN_NOVALUE]
[$DTR$FUN_NOOPTIMIZE]
$DTR$FUN_END_DEF

```

ZK-1111A-GE

A function definition contains the following parts:

\$DTR\$FUN_DEF internal-name, external-name, number-of-arguments

internal-name

Is the name used to call the function within DATATRIEVE. The internal name cannot be the same as any DATATRIEVE keyword or name.

external-name

Is the name of the library procedure or the user-written function program. It is the global symbol for the routine entry point.

number-of-arguments

Is the number of input arguments that are passed to the function. You can specify from 0 to 31 arguments. The number includes arguments that are used to return a value. The number must be the same as the number of arguments that you describe with the DTR\$FUN_IN_ARG clause.

\$DTR\$FUN_HEADER HDR = <"header-segment" [/...]>

Specifies the header that DATATRIEVE displays when you use the function as a PRINT element. The header specification must be enclosed in angle brackets. If you want no function header displayed, place a hyphen within the angle brackets (for instance, <->). If you do not specify a header, DATATRIEVE uses

Adding Functions to DATATRIEVE

4.3 How to Add Functions to DATATRIEVE

the function's internal name. The rules for formatting function headers are the same as those for formatting DATATRIEVE query headers.

\$DTR\$FUN_EDIT_STRING ^\edit-string

Specifies the edit string DATATRIEVE uses to display the function value. The edit string specification must be enclosed within the delimiters indicated. Use uppercase letters to specify the edit string.

\$DTR\$FUN_OUT_ARG TYPE = {type}, DTYPE = data-type

Tells DATATRIEVE how the value of the function is returned.

The type parameter can have the following values:

FUN\$K_STATUS

Indicates that the function value is returned in one of the input arguments and that a status code is returned in register 0. DATATRIEVE signals the status if the status code indicates a warning, an error, or a severe error. With DATATRIEVE V4.2 or later, when you define a new function you must explicitly return the status from the function code.

FUN\$K_INPUT

Indicates that the function value is returned in one of the input arguments. DATATRIEVE ignores the values returned in the registers.

FUN\$K_VALUE

Indicates that the function value is returned in registers 0 and 1. In this case, you must indicate the data type of the output argument with a DTYPE = clause. DATATRIEVE uses the data type to determine how many registers to check and how to interpret the contents of those registers. The data type for FUN\$K_VALUE can be up to 64 bits long.

DTYPE = data-type

Indicates the data type of the output argument.

Appendix C contains a complete list of the VAX data types. Table 4-1 lists the data types commonly used in functions.

\$DTR\$FUN_IN_ARG TYPE = {type} , ALL_LEN = m , OUT_PUT = TRUE, DTYPE = data-type , ORDER = p

Describes how DATATRIEVE passes input arguments. It specifies the data types of the arguments and their positions when the function is called within DATATRIEVE. This section also indicates whether an input argument is used to pass the function value to DATATRIEVE.

Adding Functions to DATATRIEVE

4.3 How to Add Functions to DATATRIEVE

There must be one `DTRFUN_IN_ARG` section for each input argument. In addition, you must specify the arguments in the order that the procedure expects to receive them. You must specify how each argument is passed with a `TYPE` clause.

You can specify the following argument-passing types with the `TYPE` parameter:

`FUN$K_NULL`

Causes `DATATRIEVE` to pass a null value. You can use this type when you want to omit an optional argument in a procedure. `FUN$K_NULL` puts an immediate value of zero in the argument list.

`FUN$K_TEXT`

Causes `DATATRIEVE` to pass a dynamic string descriptor of variable length. You can use the optional (`AL_LEN`) clause to specify the length. You cannot specify the `DTYPE` argument if the type is `FUN$K_TEXT`.

`FUN$K_VALUE`

Causes `DATATRIEVE` to pass an immediate value in the argument list. You must specify the `DTYPE` argument for this type. You can use only data types that fit into a 32-bit longword.

`FUN$K_DESC`

Causes `DATATRIEVE` to pass the address of a class S (scalar) or class SD (scalar decimal) descriptor in the argument list. You must specify the `DTYPE` clause for this type.

`FUN$K_REF`

Causes `DATATRIEVE` to pass the address of a value in the argument list. You must use the `DTYPE` clause to specify a nonstring data type for this type.

You use the following clauses with the `TYPE` parameter:

`ALL_LEN = m`

Causes `DATATRIEVE` to allocate `m` bytes to the string before calling the routine. This clause is necessary only if the external routine requires the descriptor of a fixed-length string.

The `ALL_LEN` clause is an option only with an argument whose type is `FUN$K_TEXT`.

`DTYPE = data-type`

Adding Functions to DATATRIEVE

4.3 How to Add Functions to DATATRIEVE

Indicates the data type of the input argument. If the type of the argument is FUN\$K_VALUE, FUN\$K_REF, or FUN\$K_DESC, it is necessary to specify the data type with the DTYPE clause. Do not use this clause with arguments whose type is FUN\$K_NULL or FUN\$K_TEXT.

Appendix C contains a complete list of the VAX data types. Table 4–1 lists the data types commonly used in functions.

OUT_PUT = TRUE

Specifies that the function value is passed back in the argument in which this clause appears. Only one input argument can have this clause. If the function value is returned in the registers, then no input argument should have this clause.

ORDER = p

Specifies the order of the input arguments when a user calls the function within DATATRIEVE. You must include an ORDER clause with each argument:

- Unless the passing type is FUN\$K_NULL
- Unless the argument is used for output and the passing type is FUN\$K_TEXT or FUN\$K_REF

\$DTR\$FUN_NOVALUE

Specifies that no value is returned to DATATRIEVE by the external procedure. If you use this statement, you must specify either the FUN\$K_STATUS or FUN\$K_INPUT type in the output argument. You cannot use the clause OUT_PUT = TRUE in input arguments.

The \$DTR\$FUN_NOVALUE statement enables you to invoke the function within DATATRIEVE as a command or statement rather than as a value expression.

\$DTR\$FUN_NOOPTIMIZE

Specifies that a function override the optimization that DATATRIEVE uses when it executes functions within FOR, REPEAT, or WHILE loops. For instance, DATATRIEVE optimizes the execution of a function by factoring it out of a FOR loop and executing it only once at the top of that loop. You can override how DATATRIEVE handles a function by specifying the DTR\$FUN_NOOPTIMIZE statement in the function definition. DATATRIEVE then executes the function for each execution of the FOR loop.

\$DTR\$FUN_END_DEF

Specifies the end of the function definition.

Adding Functions to DATATRIEVE

4.3 How to Add Functions to DATATRIEVE

Table 4–1 lists the data types commonly used in functions. For a complete list of the VAX data types refer to Appendix C.

Table 4–1 Common VAX Data Types

Symbol	Description
DSC\$K_DTYPE_LU	Longword unsigned. A 32-bit unsigned quantity.
DSC\$K_DTYPE_QU	Quadword unsigned. A 64-bit unsigned quantity.
DSC\$K_DTYPE_L	Longword integer. A 32-bit signed two's complement integer.
DSC\$K_DTYPE_Q	Quadword integer. A 64-bit signed two's complement integer.
DSC\$K_DTYPE_F	F_floating. A 32-bit F_floating quantity representing a single-precision number.
DSC\$K_DTYPE_D	D_floating. A 64-bit D_floating quantity representing a double-precision number.
DSC\$K_DTYPE_T	ASCII text. A string of 8-bit ASCII characters.
DSC\$K_DTYPE_ADT	Date. A 64-bit unsigned quantity.

Whether the clauses used with the type parameter are required or optional depends on the input argument type. Table 4–2 shows the required and optional clauses for each argument-passing type.

Table 4–2 Input Argument Types and Clauses

Input Argument Type	Required Clauses	Optional Clauses
FUN\$K_NULL	None	None
FUN\$K_TEXT	OUT_PUT = TRUE	ALL_LEN = m
FUN\$K_VALUE	DTYPE = data-type ORDER = p	OUT_PUT = TRUE
FUN\$K_DESC	DTYPE = data-type ORDER = p	OUT_PUT = TRUE
FUN\$K_REF (as an input argument)	DTYPE = data-type ORDER = p	None
FUN\$K_REF (as an output argument)	OUT_PUT = TRUE DTYPE = data-type	ORDER = p

Adding Functions to DATATRIEVE

4.3 How to Add Functions to DATATRIEVE

4.3.3.2 Sample Function Definitions

The following function definition allows you to round off numbers within DATATRIEVE. The actual procedure used is MTH\$JNINT, from the VMS Run-Time Library. Like most of the Run-Time Library mathematics procedures, MTH\$JNINT accepts input arguments passed by reference and returns the function value in the registers. It is a good idea to include comments with each function definition. You identify comments in the function definitions by preceding the comment text with semicolons.

```
; FN$NINT - nearest integer from floating
;
;          output is an integer
;          input is a floating point number
$DTR$FUN_DEF      FN$NINT, MTH$JNINT, 1
  $DTR$FUN_EDIT_STRING  ^\Z(9)9\
  $DTR$FUN_OUT_ARG  TYPE = FUN$K_VALUE, DTYPE = DSC$K_DTYPE_L
  $DTR$FUN_IN_ARG   TYPE = FUN$K_REF, DTYPE = DSC$K_DTYPE_F, ORDER = 1
$DTR$FUN_END_DEF
```

To use the function within DATATRIEVE, use the internal name FN\$NINT. You can use function value expressions in the same way as you use any other value expression. For example, you can use this function to round off a floating point number:

```
DTR> PRINT FN$NINT(11.2)

FN$NINT
      11
```

You can also use this function in a record selection expression:

```
DTR> FOR FIRST 3 YACHTS WITH FN$NINT(LOA/BEAM) LT 5
CON> PRINT TYPE, FN$NINT(LOA/BEAM) ("LOA/BEAM")

MANUFACTURER  MODEL      LOA/BEAM
ALBERG        37 MK II      3
ALBIN         79              2
ALBIN         BALLAD    3
```

The following function definition uses an input argument to pass back the function value:

Adding Functions to DATATRIEVE

4.3 How to Add Functions to DATATRIEVE

```
; FN$STR_EXTRACT - String extract
;
;     output is a string
;     input is an output string descriptor,
;     an input string descriptor
;     a starting position in the string
;     and the length to extract
$DTR$FUN_DEF FN$STR_EXTRACT, STR$LEN_EXTR, 4
    $DTR$FUN_HEADER HDR = <"Extracted"/"String">
    $DTR$FUN_OUT_ARG TYPE = FUN$K_STATUS
    $DTR$FUN_IN_ARG  TYPE = FUN$K_TEXT , OUT_PUT = TRUE
    $DTR$FUN_IN_ARG  TYPE = FUN$K_DESC, DTYPE = DSC$K_DTYPE_T, ORDER = 1
    $DTR$FUN_IN_ARG  TYPE = FUN$K_REF, DTYPE = DSC$K_DTYPE_L, ORDER = 2
    $DTR$FUN_IN_ARG  TYPE = FUN$K_REF, DTYPE = DSC$K_DTYPE_L, ORDER = 3
$DTR$FUN_END_DEF
```

When you call this function within DATATRIEVE, the first input argument you supply is the string from which you want to extract a substring. This corresponds to the argument with the ORDER = 1 clause. The second and third arguments are the start position and the length of the substring. The fact that the actual procedure uses an input argument to return a value is invisible to a user calling the function with DATATRIEVE. You can use FN\$STR_EXTRACT as follows:

```
DTR> FOR FIRST 3 YACHTS PRINT FN$STR_EXTRACT(BUILDER, 3, 4)
```

```
    Extracted
    String
```

```
BERG
BIN
BIN
```

The next examples are the definitions of the functions FN\$INIT_TIMER and FN\$SHOW_TIMER:

```
; FN$INIT_TIMER - Initializes a timer and counter
;
;     No Output
$DTR$FUN_DEF FN$INIT_TIMER, LIB$INIT_TIMER, 0
    $DTR$FUN_OUT_ARG TYPE = FUN$K_STATUS
    $DTR$FUN_NOVALUE
$DTR$FUN_END_DEF

; FN$SHOW_TIMER - Displays elapsed time, CPU time and
;                 other process parameters
;     No Output
$DTR$FUN_DEF FN$SHOW_TIMER, LIB$SHOW_TIMER, 0
    $DTR$FUN_OUT_ARG TYPE = FUN$K_STATUS
    $DTR$FUN_NOVALUE
$DTR$FUN_END_DEF
```

Adding Functions to DATATRIEVE

4.3 How to Add Functions to DATATRIEVE

The `DTRFUN_NOVALUE` statement in these function definitions enables you to invoke them as DATATRIEVE commands. You can use these functions to measure the performance of DATATRIEVE in response to various queries, as in the following example:

```
DTR> SHOW TIMER
PROCEDURE TIMER
READY YACHTS
FN$INIT_TIMER
FIND YACHTS WITH BUILDER = 'ALBIN'
PRINT CURRENT
FN$SHOW_TIMER
RELEASE CURRENT
FN$INIT_TIMER
PRINT YACHTS WITH BUILDER = 'ALBIN'
FN$SHOW_TIMER
END_PROCEDURE
DTR> :TIMER
```

MANUFACTURER	MODEL	RIG	LENGTH		WEIGHT	BEAM	PRICE
			ALL	OVER			
ALBIN	79	SLOOP	26		4,200	10	\$17,900
ALBIN	BALLAD	SLOOP	30		7,276	10	\$27,500
ALBIN	VEGA	SLOOP	27		5,070	08	\$18,600
ELAPSED: 00:00:00.60			CPU: 0:00:00.20		BUFIO: 8	DIRIO: 3	FAULTS: 31

MANUFACTURER	MODEL	RIG	LENGTH		WEIGHT	BEAM	PRICE
			ALL	OVER			
ALBIN	79	SLOOP	26		4,200	10	\$17,900
ALBIN	BALLAD	SLOOP	30		7,276	10	\$27,500
ALBIN	VEGA	SLOOP	27		5,070	08	\$18,600
ELAPSED: 00:00:00.41			CPU: 0:00:00.16		BUFIO: 8	DIRIO: 1	FAULTS: 0

The following example is the function definition for the VAX BASIC procedure `POWER` described in Section 4.3.1, *Write and Compile Your Procedures*, earlier in this chapter:

```
; FN$POWER - uses first input as the base, second input as exponent
;
;      output is floating number
;      input is two floating numbers
$DTR$FUN_DEF FN$POWER, POWER, 2
  $DTR$FUN_OUT_ARG  TYPE = FUN$K_VALUE, DTYPE = DSC$K_DTYPE_F
  $DTR$FUN_IN_ARG   TYPE = FUN$K_REF, DTYPE = DSC$K_DTYPE_F, ORDER = 1
  $DTR$FUN_IN_ARG   TYPE = FUN$K_REF, DTYPE = DSC$K_DTYPE_F, ORDER = 2
$DTR$FUN_END_DEF
```

Adding Functions to DATATRIEVE

4.3 How to Add Functions to DATATRIEVE

You can call the external procedure `POWER` from within `DATATRIEVE` using the internal name `FN$POWER`. `FN$POWER` accepts a number and its exponent, and returns the value of the number to the specified exponent.

```
DTR> FIND YACHTS WITH LOA GT 40; SELECT 1
DTR> PRINT LOA, FN$POWER(LOA,2) ("LOA"/"SQUARED") USING Z(5)V99
LENGTH
OVER   LOA
ALL    SQUARED
41     1681.00
```

The following example is the function definition for the VAX FORTRAN procedure `CLEAR` described in Section 4.3.1, *Write and Compile Your Procedures*, earlier in this chapter:

```
; FN$CLEAR - Clears the screen
;
;      output is an escape sequence
;      input is an output string descriptor
$DTR$FUN_DEF  FN$CLEAR, CLEAR, 1
  $DTR$FUN_HEADER  HDR = <->
  $DTR$FUN_OUT_ARG TYPE = FUN$K_STATUS
  $DTR$FUN_IN_ARG  TYPE = FUN$K_TEXT, ALL_LEN = 7, OUT_PUT = TRUE
$DTR$FUN_END_DEF
```

You can use `FN$CLEAR` to clear a VT100 terminal screen as follows:

```
DTR> PRINT FN$CLEAR
```

4.3.4 Assemble and Debug `DTRFNDxx.MAR`

After you place function definitions in `DTRFNDxx.MAR`, reassemble it with the command:

```
$ MACRO/LIST DTR$LIBRARY:DTRFNDxx.MAR
```

If you have made any mistakes in your function definitions, you should get error messages when you assemble `DTRFNDxx.MAR`. The following table contains explanations of these error messages:

Error 1: Function definition not in progress.
 Occurs if you omit the `DTRFUN_DEF` section from your function definition.

Adding Functions to DATATRIEVE

4.3 How to Add Functions to DATATRIEVE

- Error 2:** Duplicate definition within a single function.
Occurs if:
- You specify `DTRFUN_DEF` without a corresponding `DTRFUN_END_DEF`
 - You have two `DTROUT_ARG` sections in a function definition
 - You specify two arguments as `OUT_PUT = TRUE`
- Error 3:** Parameter out of range.
Occurs if:
- You have a data type or passing type that is not in the acceptable range for an argument
 - You specify more than 31 input arguments
 - You specify an allocation length greater than 1000
- Error 4:** Calling order is not dense.
Occurs if:
- You specify input arguments in an incorrect order. For example, if you have three input arguments and you use the clauses `ORDER = 1`, `ORDER = 3`, and `ORDER = 4`, then the calling order is not dense.
 - You have fewer input arguments than you specify in the `DTRFUN_DEF` section.
- Error 5:** More input arguments than specified in function definition.
Occurs if you include more input arguments than you specify in the `DTRFUN_DEF` section.
- Error 6:** Order must be specified for this passing type.
Occurs if you do not include the `ORDER =` clause when it must be specified. Each `DTRFUN_IN_ARG` must have an `ORDER =` clause unless the passing type is `FUN$K_NULL` or unless you include an `OUTPUT = TRUE` clause and the passing type is `FUN$K_TEXT` or `FUN$K_REF`.
- Error 7:** Duplicate call order specified.
Occurs if you specify the same order for two input arguments.
- Error 8:** Output argument not specified.
Occurs if you leave out the `DTRFUN_OUT_ARG` section.

Adding Functions to DATATRIEVE

4.3 How to Add Functions to DATATRIEVE

Error 9: Output argument not found in input arguments.

Occurs if you do not specify the `FUN$K_VALUE` passing type for `DTRFUN_OUT_ARG`, and you also do not have any `DTRFUN_IN_ARG` sections with the `OUT_PUT = TRUE` clause.

If you get any of these messages when you assemble `DTRFNDxx.MAR`, you should note the lines that caused the errors. You can then determine which function definitions have errors and fix them. (If you use the `/LIST` qualifier with the `MACRO` command, you can also use the `.LIS` file that the `MACRO` assembler creates to help you identify the incorrect lines.)

4.3.5 Replace `DTRFNDxx.MAR` in the Object Library

After you successfully assemble `DTRFNDxx.MAR`, use the `LIBRARY` command to put the new object file, `DTRFNDxx.OBJ`, in the library `DTRFUNxx.OLB`:

```
$ LIBRARY/REPLACE DTR$LIBRARY:DTRFUNxx DTRFNDxx
```

4.3.6 Relink the `DATATRIEVE` Shareable Image

After you replace `DTRFNDxx.OBJ` in the `DATATRIEVE` object module library, you must relink the `DATATRIEVE` shareable image for the new function definitions to take effect.

You use the command procedure `DTRBLDxx.COM` to relink the `DATATRIEVE` shareable image. When you relink, you must be sure that the shareable image you relink against is from the most recently installed version of `DATATRIEVE`.

Before invoking `DTRBLDxx.COM`, check the command file to make sure it includes all the Run-Time Libraries that your function definitions require. By default, `DATATRIEVE` links against only those Run-Time Libraries (RTLs) that it requires. For example, `DATATRIEVE` does not automatically link against the `BASIC` Run-Time Library.

When you write a function definition that uses an RTL routine in an object library that `DATATRIEVE` does not link against, you must edit `DTRBLDxx.COM` to include that RTL library. For example, to use the `BASIC` procedure `POWER` described in Section 4.3.1, `Write and Compile Your Procedures`, and defined in Section 4.3.3.2, `Sample Function Definitions`, you must add these two lines to the `DTRBLDxx.COM` file:

```
CLUSTER=BASRTL, , ,SYS$COMMON: [SYSLIB]BASRTL.EXE/SHAREABLE  
CLUSTER=BASRTL2, , ,SYS$COMMON: [SYSLIB]BASRTL2.EXE/SHAREABLE
```

Put the lines for the `BASIC` RTL in the same section in the command file that you find the following lines:

```
CLUSTER=FORRTL, , ,SYS$COMMON: [SYSLIB]FORRTL.EXE/SHAREABLE  
CLUSTER=LIBRTL, , ,SYS$COMMON: [SYSLIB]LIBRTL.EXE/SHAREABLE
```


Adding Functions to DATATRIEVE

4.3 How to Add Functions to DATATRIEVE

To relink VAX DATATRIEVE, execute the command file DTRBLDxx:

```
$ @DTR$LIBRARY:DTRBLDxx
```


5

Customizing DATATRIEVE Help Text

You can customize any DATATRIEVE help text and make additions to the existing text. Changes and additions you make can become part of your own DATATRIEVE on-line help. If you have the privilege SYSPRV, you can also change and add help text for all DATATRIEVE users on your system.

This chapter explains how to modify or add DATATRIEVE help text to suit your on-line documentation needs.

5.1 How to Change Help Text

To change DATATRIEVE help text, follow these steps:

1. Copy the DATATRIEVE help library file DTRHELP.HLB from SYS\$HELP to one of your own directories.
2. Extract the help modules you want to change.
3. Edit the extracted help text to make the desired changes.
4. Replace the changed text in DTRHELP.HLB.
5. Replace the old help library file with the new one.

The following sections describe each of these steps.

5.1.1 Copy the DATATRIEVE Help Library File

The DATATRIEVE help library file, DTRHELP.HLB, is located in the SYS\$HELP directory. To copy DTRHELP.HLB to one of your own directories, use the COPY command. For example:

```
$ COPY SYS$HELP:DTRHELP.HLB DB0:[USER.DTR]DTRHELP.HLB
```

This command creates a copy of DTRHELP.HLB in the DB0:[USER.DTR] directory.

Customizing DATATRIEVE Help Text

5.1 How to Change Help Text

5.1.2 Extract the Help Modules You Want to Change

The help library file DTRHELP.HLB consists of text modules of each DATATRIEVE help topic. To see what modules DTRHELP.HLB contains, use the LIBRARY command with the /LIST and /HELP qualifiers:

```
$ LIBRARY/HELP/LIST DTRHELP.HLB
```

After you copy DTRHELP.HLB to your own directory, use the LIBRARY command to extract the modules you wish to change. For example, if you want to modify the help text for the DATATRIEVE FUNCTIONS command, use the following LIBRARY command:

```
$ LIBRARY/HELP/EXTRACT=FUNCTIONS/OUTPUT=FUNCTIONS DTRHELP.HLB
```

The /EXTRACT qualifier causes the VMS Librarian Utility to extract the FUNCTIONS module from DTRHELP.HLB. The /OUTPUT qualifier places the HELP FUNCTIONS text in the file FUNCTIONS.HLP in your default VMS directory. Note that FUNCTIONS.HLP, the file that results from the LIBRARY command, contains text.

For more information on the LIBRARY command, use the HELP LIBRARY command or refer to the VMS documentation set.

5.1.3 Edit the Extracted Help Text

After you extract the module you want to change, use an editor to make changes to the .HLP text file.

5.1.3.1 The Structure of Help Text Files

Help files have a hierarchical structure, much like DATATRIEVE record definitions.

Each numbered word in the file is a key. The top-level key is the keyword FUNCTIONS. When you enter the command HELP FUNCTIONS in response to the DTR> prompt, DATATRIEVE displays the keyword FUNCTIONS and text that explains its use. In addition, DATATRIEVE displays the subtopics you can get further help on. The FUNCTIONS subtopics are the second-level keys, that is, those words preceded by the number 2.

5.1.4 Replace the Changed Text in DTRHELP.HLB

After you make the desired changes to the help text, use the LIBRARY command with the /REPLACE qualifier to replace the old help module with your new one. Use the following command to replace the FUNCTIONS module in DTRHELP.HLB with the changed text in FUNCTIONS.HLP:

Customizing DATATRIEVE Help Text

5.1 How to Change Help Text

```
$ LIBRARY/HELP/REPLACE
_Library:      DTRHELP
_File:         FUNCTIONS.HLP
```

5.1.5 Replace the Old Help Library File with the New One

If you want all DATATRIEVE users on your system to be able to display your new help text, copy the new DTRHELP.HLB file to SYS\$HELP. You must have write access to SYS\$HELP to do this. Be sure to use the SET PROTECTION command to give READ privilege to world category users.

If you want only selected users to be able to display your new help text, assign a process logical name that refers to the new help library file. For example, if the new DTRHELP.HLB is in DB0:[USER.DTR], use the following command:

```
$ ASSIGN "DB0:[USER.DTR]DTRHELP" DTRHELP
```

Users who want DATATRIEVE to display the changed help text can include this command in their LOGIN.COM file.

You can also use the function FN\$CREATE_LOG to define the logical name DTRHELP after you enter DATATRIEVE:

```
DTR> FN$CREATE_LOG ("DTRHELP", "DB0:[USER.DTR]DTRHELP")
```

The FN\$CREATE_LOG assignment is valid only during the current DATATRIEVE session. You can make this assignment automatic by inserting the FN\$CREATE_LOG command in a DATATRIEVE startup command file.

5.2 How to Create New Help Text

The procedure for creating new help text is similar to the procedure for modifying existing help text. Design your help files with the hierarchical structure described in this chapter. Use an editor to create the help files. You can use DIGITAL Standard Runoff to format your help text if you wish. Be sure to place each numbered keyword at the left-hand margin.

After you create your new help text, use the LIBRARY command with the /INSERT qualifier to insert the new text into DTRHELP.HLB. If your help text file is HELPTEXT.HLP, you can make it part of the help library with this command:

```
$ LIBRARY/HELP/INSERT
_Library:      DTRHELP
_File:         HELPTEXT
```

To have DATATRIEVE display the new help text, follow the procedure described in the preceding section.

6

Customizing DATATRIEVE Messages

This chapter explains how you can change the text of the DATATRIEVE messages to suit the needs of your working environment. You can change the messages DATATRIEVE displays to you and to other selected users. If you have the privilege SYSPRV, you can modify messages for all DATATRIEVE users on your system.

This manual does not describe how to change messages from other products such as VAX/VMS, VAX RMS, VAX CDD/Repository, VAX DBMS, and the VMS DIGITAL Standard Editor (EDT).

6.1 DATATRIEVE Messages

Each time you give DATATRIEVE a command or statement to execute, DATATRIEVE determines the message it should display. Each message includes a severity level, a code, a name, and a text string.

The message **severity level** determines whether DATATRIEVE completes your command or statement and what warning or information DATATRIEVE displays. Your command or statement results in one of the following severity levels:

- Severe Error
- Error
- Warning
- Informational

The message **code** is a number that identifies the message.

The message **name** is a symbol beginning with the prefix DTR\$_ that also identifies the message. The message code and the message name are synonymous; they represent the same value. In most cases, the DATATRIEVE documentation recommends using the message name rather than the actual hexadecimal code value because the name is easier to remember.

The message **text** is a string that DATATRIEVE displays to tell you how it is responding to your command or statement.

Customizing DATATRIEVE Messages

6.1 DATATRIEVE Messages

A list describing the severity level, code, name, and text for all DATATRIEVE messages is supplied on-line as part of the DATATRIEVE installation procedure. See Appendix B for information on how to access the on-line message documentation.

6.1.1 DATATRIEVE Messages and the Call Interface

When a calling program passes DATATRIEVE a command or statement, DATATRIEVE returns a message. For example, if DATATRIEVE successfully executes your command or statement, it returns the following information:

```
Severity level:      Informational
Code:               008D8563
Name:              DTR$_SUCCESS
Text:              Statement completed successfully.
```

Your program can determine the severity level, code, and message text by examining the contents of the DAB\$L_CONDITION and DAB\$A_MSG_BUF fields of the DATATRIEVE Access Block after each call. Your program can use the message names in place of the code values by declaring the names as external constants and linking with the DATATRIEVE shareable image. See Chapter 2 for more information on handling DATATRIEVE messages in a calling program.

6.1.2 Messages in Interactive DATATRIEVE

When you enter a command or statement using interactive DATATRIEVE, DATATRIEVE does not always display a message. For example, if you complete a command or statement successfully in interactive DATATRIEVE, you do not see the DTR\$_SUCCESS message.

However, in many situations, interactive DATATRIEVE does display messages in response to your commands and statements. Here is an example:

```
DTR> READY PERSONEL
Element "PERSONEL" not found in dictionary.      1
DTR> READY
[Looking for dictionary path name]              2
CON> PERSONNEL AS PERS
DTR> PRINT PERS WITH DEPT = D98
"D98" not field, assumed literal.              3
```

ID	STATUS	FIRST NAME	LAST NAME	DEPT	START DATE	SALARY	SUP ID
02943	EXPERIENCED	CASS	TERRY	D98	2-Jan-1980	\$29,908	39485
39485	EXPERIENCED	DEE	TERRICK	D98	2-May-1977	\$55,829	00012
49843	TRAINEE	BART	HAMMER	D98	4-Aug-1981	\$26,392	39485
84375	EXPERIENCED	MARY	NALEVO	D98	3-Jan-1976	\$56,847	39485

Customizing DATATRIEVE Messages

6.1 DATATRIEVE Messages

DTR>

Table 6–1 lists the severity level, code, name, and text for each message in this example:

Table 6–1 Sample DATATRIEVE Messages

Example Number	Severity Level	Code	Name	Text
1	Error	008D8192	DTR\$_ELTNOTDIC	Element < . . . > not found in dictionary.
2	Informational	008D857B	DTR\$_LOOKINFO	[Looking for < . . . >]
3	Warning	008D84E0	DTR\$_ASSUMELIT	< . . . > not field, assumed literal.

You cannot change the severity level, code, or name of any message. However, you can change the message text. The following sections describe how to do this.

6.2 How to Change DATATRIEVE Messages

To change DATATRIEVE messages, you must perform the following steps:

1. Copy the DATATRIEVE message source file DTRMSG.SMSG from DTR\$LIBRARY to your own directory.
2. Edit the message source file to make the desired changes.
3. Compile the message source file to create an object file.
4. Link the object file to create the file DTRMSG.EXE.
5. Replace the old DTRMSG.EXE file with the new one.

The following sections describe each of these steps.

6.2.1 Copy the Message Source File

The message source file DTRMSG.SMSG is located in DTR\$LIBRARY. Use the COPY command to copy this file to one of your own directories. For example:

```
$ COPY DTR$LIBRARY:DTRMSG.SMSG DB3:[CASSIDY.WORK]DTRMSG.SMSG
```

This command creates a copy of DTRMSG.SMSG in the DB3:[CASSIDY.WORK] directory.

Customizing DATATRIEVE Messages

6.2 How to Change DATATRIEVE Messages

6.2.2 Edit the Message Source File

After you copy DTRMSG.SMSG, use an editor, such as EDT, to make the desired changes to the message text. The following sections describe the structure of DTRMSG.SMSG and explain how to edit it.

6.2.2.1 Structure of the Message Source File

DTRMSG.SMSG, the message source file, contains information that defines each DATATRIEVE message.

At the top of the file is a .TITLE directive that identifies the message source file. Following the title is the .FACILITY statement that identifies the software product. The /SYSTEM qualifier means that DATATRIEVE is a facility supplied by DIGITAL. The /PREFIX qualifier indicates that each message name has a DTR\$_ prefix.

Following the facility definition is the number specifier and definition for each DATATRIEVE message.

The .BASE **message number specifier** is a value used to calculate the message code. You should not modify this number.

After each message number specifier is a **message definition**. The message definition has the following format:

**name /FAO=n/severity-level -
<message-text>**

name

Is a text string that is combined with the DTR\$_ prefix to make up the message name.

/FAO=n

Specifies the number of formatted ASCII output (FAO) arguments to be included in the message. See Section 6.2.2.3 for more information on FAO arguments.

/severity-level

Indicates the severity level associated with the message.

message-text

Is the message text that DATATRIEVE displays. The text must be on one line and it can be up to 255 bytes long. The delimiters of the text can be angle brackets (< >) or double quotation marks (" "). The text can include FAO directives that insert ASCII strings into the resulting message. See Section 6.2.2.3 for more information on FAO directives.

Customizing DATATRIEVE Messages

6.2 How to Change DATATRIEVE Messages

The only part of the message source file that you should change is the message text. If you change any other part of the file, DATATRIEVE may not display error messages correctly.

The following section explains how to change the message text.

6.2.2.2 Changing the Message Text

The following example shows a message you may want to change:

```
DTR> READY PERSONNEL
Bad record size. Defined: 43 File: 41
DTR>
```

The error in this example, DTR\$_BADRECSIZ, occurs when a user attempts to ready a domain whose record definition specifies a size that does not match the size of the record in the data file.

The message definition for DTR\$_BADRECSIZ in DTRMSG.SMSG is as follows:

```
BADRECSIZ/FAO=2/ERROR-
<Bad record size. Defined: !UW File: !UW >
```

Suppose you want to supply more information about why the READY statement in this example failed. You want DATATRIEVE to display the following message:

```
DTR> READY PERSONNEL

Your record definition describes each record as 43 bytes long.
In your file each record is 41 bytes long.

DTR>
```

To cause DATATRIEVE to display this new message, edit DTRMSG.SMSG and change the message definition as follows:

```
BADRECSIZ/FAO=2/ERROR-
<!/Your record definition describes each record as !UW bytes long.
!/In your file each record is !UW bytes long.!/>
```

You should not alter the name or the FAO=*n* and severity level qualifiers of messages you modify. You should edit only the message text. Note that the message text must be on one line with a maximum size of 255 characters. Do not include line feeds or carriage returns in the message text.

The new message text in DTRMSG.SMSG contains characters preceded by an exclamation mark (!) such as !UW and !/. These are FAO directives. The following section explains how to use FAO directives to format message texts.

Customizing DATATRIEVE Messages

6.2 How to Change DATATRIEVE Messages

6.2.2.3 Using FAO Directives to Format Message Text

DATATRIEVE uses the Formatted ASCII Output (FAO) system service to:

- Insert character string data into message texts
- Convert binary values into the ASCII representation of their decimal or hexadecimal values and substitute the values into the message text
- Format the message text

See the VMS documentation set for information on the FAO service. Table 6–2 summarizes the FAO directives used in DATATRIEVE message texts and lists the number of parameters required by each directive.

Table 6–2 FAO Directives Used in DATATRIEVE Messages

Directive	Function	Number of Parameters
Substitution		
!AC	Inserts a counted ASCII string.	1
!AD	Inserts an ASCII string.	2
!AF	Inserts an ASCII string; replaces nonprintable codes with periods (.).	2
!AS	Inserts an ASCII string.	1
!SL	Converts a signed decimal longword.	1
!UL	Converts an unsigned decimal longword.	1
!UW	Converts an unsigned decimal word.	1
!XL	Converts a longword to hexadecimal.	1

(continued on next page)

Customizing DATATRIEVE Messages

6.2 How to Change DATATRIEVE Messages

Table 6–2 (Cont.) FAO Directives Used in DATATRIEVE Messages

Directive	Function	Number of Parameters
Output String Formatting		
!/	Inserts a new line (<CR> and <LF>).	None
!_	Inserts a tab.	None
!^	Inserts a form feed.	None
!!	Inserts an exclamation point.	None
!%S	Inserts the letter S if the most recently converted numeric value is not 1.	None
!n< !>	Defines an output field width of n characters. All data within delimiters is left justified and blank filled. The outer delimiters of the message text should be changed to " ".	None
!n*c	Repeats the specified character c, n times.	None
Parameter Interpretation		
!-	Reuses the last parameter in the list.	None
!+	Skips the next parameter in the list.	None

The following sections describe how to use FAO directives.

6.2.2.3.1 Substitution Directives The following example shows how DATATRIEVE uses two FAO substitution directives:

```
DTR> FIND
[Looking for name of domain, collection, or list]
CON> FIRST 10 PERSONNEL
[10 records found]

DTR>
```

The statement in this example results in two messages. The first is DTR\$_LOOKINFO. This message has the following definition:

```
LOOKINFO/FAO=1/INFORMATIONAL-
<[Looking for !AC] >
```

Customizing DATATRIEVE Messages

6.2 How to Change DATATRIEVE Messages

DATATRIEVE substitutes the string “name of domain, collection, or list” for the directive !AC. If you want to modify this or other variable strings that DATATRIEVE substitutes, you must edit the file DTRTEXT.MAR. See Chapter 8 for information about modifying DATATRIEVE text.

The second message in the example is DTR\$_RECFOUND. DTR\$_RECFOUND has the following definition:

```
RECFOUND/FAO=1/INFORMATIONAL-  
<[!UL record!%S found] >
```

DATATRIEVE substitutes the number of records found for the directive !UL. In addition, the !%S formatting directive causes DATATRIEVE to insert an “s” at the end of “record” when more than one record is found.

You can delete substitution directives and cause DATATRIEVE not to perform the substitution. If there are more than one substitution directives, you can change their order, as described in Section 6.2.2.3.3, you should not replace one substitution directive with another or insert a substitution directive into message text. Replacing or inserting substitution directives can cause errors in the messages DATATRIEVE displays.

6.2.2.3.2 Formatting Directives While you should not change substitution directives, you can use formatting directives to reformat messages.

The following example shows how you can reformat a message:

```
DTR> PRINT AVERAGE PRICE OF YACHTS WITH RIG = "SLOOP"  
  
AVERAGE  
PRICE  
  
[Function computed using 38 of 96 values.]  
$20,464  
  
DTR>
```

The informational message in this example is DTR\$_STAMISDAT; it has the following definition:

```
STAMISDAT/FAO=2/INFORMATIONAL-  
<[Function computed using !UL of !UL values.] >
```

When DATATRIEVE displays this message, it substitutes the number of YACHTS records that RIG = “SLOOP” groups together in the second !UL directive. It substitutes the number of YACHTS whose price is not missing in the first directive.

Customizing DATATRIEVE Messages

6.2 How to Change DATATRIEVE Messages

Suppose you want to reformat the DTR\$_STAMISDAT message so that DATATRIEVE displays it as follows:

```
DTR> PRINT AVERAGE PRICE OF YACHTS WITH RIG = "SLOOP"
AVERAGE
  PRICE
          This function was computed using 38 of 96 values.
          Values not used are missing.
$20,464
DTR>
```

To make this change, edit DTRMSG.MSG and use the !_ and !/ formatting directives to insert tabs and new lines. Be sure that the message text is on one line:

```
<!_!_This function was computed using !UL of !UL values.!/_!_Values
not used are missing.!/ >
```

6.2.2.3.3 Parameter Interpretation Directives You can use parameter interpretation directives to change the order of substitution directives. For example, the following message contains two string substitution directives:

```
DTR> PRINT EMPLOYEES WITH LOCATION = "MK"
PRIMT EMPLOYEES WITH LOCATION = "MK"
^
Expected statement, encountered "PRIMT".
DTR>
```

The mistyped word in this example causes the error DTR\$_SYNTAX, which has following message definition:

```
SYNTAX/FAO=3/ERROR-
<Expected !AC, encountered "!AD". >
```

When DATATRIEVE displays the error message, it substitutes the string “statement” for !AC and the string “PRIMT” for !AD. Note that the FAO=3 qualifier specifies the total number of parameters for all directives in the message text. Remember that !AD takes two parameters.

Suppose you want to reformat the DTR\$_SYNTAX error message. You want to change the dialogue in the previous example to:

```
DTR> PRINT EMPLOYEES WITH LOCATION = "MK"
PRIMT EMPLOYEES WITH LOCATION = "MK"
^
You entered "PRIMT".
I was expecting a statement . . . Try again.
DTR>
```

Customizing DATATRIEVE Messages

6.2 How to Change DATATRIEVE Messages

To make this change, use an editor to edit the current message text in DTRMSG.SMSG:

```
<Expected !AC, encountered "!AD". >
```

Change this text to:

```
<You entered !+"!AD".!/I was expecting a !-!-!-!AC!3*. Try again. >
```

In this example, the parameter interpretation directives !+ and !- are used to indicate the reversal of the order of the substitution directives. In the original message definition, DATATRIEVE substitutes one parameter for !AC and then two for !AD.

In the new message definition, the !+ directive causes DATATRIEVE to skip the first parameter and substitute the next two in !AD. The three !- directives cause DATATRIEVE to back up to the first parameter and substitute it in !AC.

In addition to the parameter interpretation directives, this example contains two formatting directives. The !/ directive inserts a new line, and the !3*. directive inserts three periods (. . .).

6.2.3 Compile the Message Source File

After you edit DTRMSG.SMSG to make the desired changes, use the MESSAGE command to compile the message source file:

```
$ MESSAGE DTRMSG.SMSG
```

This command creates a DTRMSG.OBJ file in your default directory.

For more information about the MESSAGE command, enter HELP MESSAGE or refer to the VMS documentation set.

6.2.4 Link the Object File

After you create an object file, use the LINK command to create the file DTRMSG.EXE:

```
$ LINK/SHAREABLE=DTRMSG.EXE DTRMSG.OBJ
```

The DTRMSG.EXE file that the LINK command creates is a nonexecutable message file.

Customizing DATATRIEVE Messages

6.2 How to Change DATATRIEVE Messages

6.2.5 Replace the Old DTRMSG.S.EXE File with the New One

If you want DATATRIEVE to display the changed messages to all users on your system, copy the new DTRMSG.S.EXE message file to SYS\$COMMON. You must have write access to SYS\$COMMON to do this. Use the command:

```
$ COPY DTRMSG.S.EXE SYS$COMMON:DTRMSG.S.EXE
```

If you want DATATRIEVE to display the changed messages only to selected users, assign the process logical DTRMSG.S to the new DTRMSG.S.EXE file for each of these users. For example, if the new DTRMSG.S.EXE is in the DB3:[CASSIDY.WORK] directory, use the following command:

```
$ ASSIGN "DB3:[CASSIDY.WORK]DTRMSG.S.EXE" DTRMSG.S
```

Users who want DATATRIEVE to display the changed messages can include this command in their LOGIN.COM file.

Customizing ADT and Guide Mode

ADT (Application Design Tool) is a DATATRIEVE utility that helps users define domains, records, and files. Operating interactively, ADT guides the user through the process of defining data and creating data files. In addition, ADT lets the user insert domain and record definitions in the dictionary.

To use ADT, enter the following command at the DATATRIEVE prompt:

```
DTR> ADT
```

Guide Mode is a DATATRIEVE utility that guides you through a DATATRIEVE session with a series of prompts. In Guide Mode, you can get a list of all the valid commands, statements, names, or value expressions that can be entered at the point the list is requested.

Guide Mode is particularly useful to the new user who wants assistance during a DATATRIEVE session. To try a session in Guide Mode, enter the following command:

```
DTR> SET GUIDE
```

This chapter explains how you can customize ADT and Guide Mode to suit the needs of your working environment.

7.1 Customizing ADT

ADT displays a series of prompts and provides help text to aid users in defining data and creating data files. To change these prompts or help text, follow these steps:

1. Copy the ADT text file to one of your own directories.
2. Use DATATRIEVE to modify or add ADT text.
3. Replace the old ADT text file with the new one.

The following sections describe these steps.

Customizing ADT and Guide Mode

7.1 Customizing ADT

7.1.1 Copy the ADT Text File

The ADT text file, DTRADT.DAT, is located in SYS\$LIBRARY. Copy it to one of your directories, keeping the name DTRADT.DAT; for example:

```
$ COPY SYS$LIBRARY:DTRADT.DAT DB0:[CASSIDY.DTR]DTRADT.DAT
```

7.1.2 Modify ADT Text

Before you begin to modify ADT text, you should have enough experience with ADT to know what ADT text elements you want to change. Note that ADT offers you two sets of prompts to guide you through a session: brief and detailed.

All ADT prompts and help text are in the file DTRADT.DAT. You can modify all text by using the DATATRIEVE domain ADT_TEXT.

7.1.2.1 Ready the ADT_TEXT Domain

After you copy DTRADT.DAT to one of your directories, invoke DATATRIEVE from that directory. Set your default dictionary directory to CDD\$TOP.DTR\$LIB.ADT and ready the domain ADT_TEXT:

```
DTR> SET DICTIONARY CDD$TOP.DTR$LIB.ADT
DTR> READY ADT_TEXT MODIFY
```

ADT_TEXT has the following definition:

```
DOMAIN ADT_TEXT USING ADT_TEXT_REC ON DTRADT.DAT;
```

ADT_TEXT has the following record definition:

```
RECORD ADT_TEXT_REC USING
01      KEYED_TEXT.
        05 ACCESS_KEY WORD.
        05 TEXT_STRING PIC X(80).
;
```

7.1.2.2 Modify ADT Text Strings

You can change any ADT text line using DATATRIEVE and the ADT_TEXT domain. For example, the first ADT prompt is as follows:

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

If you respond with NO, ADT displays its brief prompt:

Enter domain name :

If you type YES, ADT displays its detailed prompt:

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

Customizing ADT and Guide Mode

7.1 Customizing ADT

The following example shows one way to modify the detailed prompt.

```
DTR> PRINT ADT_TEXT WITH TEXT_STRING CONT "for your domain"
```

```
ACCESS          TEXT
KEY             STRING
```

```
6146
```

```
Enter the name for your domain.
```

```
DTR> ! Use the ACCESS_KEY to find all lines of this prompt:
```

```
DTR> FIND ADT_TEXT WITH ACCESS_KEY = 6146
```

```
[4 records found]
```

```
DTR> PRINT ALL TEXT_STRING
```

```
TEXT
STRING
```

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

```
DTR> SELECT 4
```

```
DTR> MODIFY TEXT_STRING
```

```
Enter TEXT_STRING: Do not use spaces or tabs :
```

```
DTR> PRINT TEXT_STRING
```

```
TEXT
STRING
```

```
Do not use spaces or tabs :
```

```
DTR>
```

You should modify only the TEXT_STRING field of ADT_TEXT records. Changing the ACCESS_KEY field can result in errors during ADT sessions.

Some of the text strings contain FAO directives such as !+ and !AC. These directives are used to format ADT text and to substitute text strings in ADT text lines. You can use FAO directives to format ADT text in the same way you use these directives to format DATATRIVEE messages. For information about using FAO directives, refer to Chapter 6 and to the VMS documentation set.

7.1.2.3 Add ADT Text Strings

You can add to the ADT prompts and help text. For example, when ADT prompts for a file name, entering a question mark produces the following help text:

```
DATATRIVEE uses this file name in the DEFINE DOMAIN command
to find your data for the domain
```

Customizing ADT and Guide Mode

7.1 Customizing ADT

The following example shows how you can modify and add to this text:

```
DTR> READY ADT_TEXT WRITE
DTR> FIND THIS IN ADT_TEXT WITH TEXT_STRING CONT
CON> "uses this file name"
[1 record found]
DTR> SELECT
DTR> LIST THIS.ACCESS_KEY

ACCESS_KEY :    3843

DTR> FOR ADT_TEXT WITH ACCESS_KEY = THIS.ACCESS_KEY
CON> BEGIN
CON>     PRINT SKIP, TEXT_STRING(-), SKIP
CON>     MODIFY TEXT_STRING
CON> END
```

DATATRIEVE uses this file name in the DEFINE DOMAIN command

Enter TEXT_STRING: DATATRIEVE creates a file with the name you specify.

to find your data for the domain.

Enter TEXT_STRING: For example, if you enter EMPLOYEE,

```
DTR> REPEAT 2 STORE ADT_TEXT USING
CON> BEGIN
CON>     ACCESS_KEY = THIS.ACCESS_KEY
CON>     TEXT_STRING = *."new text"
CON> END
```

Enter new text: DATATRIEVE creates a file named EMPLOYEE.DAT.

Enter new text: DATATRIEVE uses this file to store your data.

```
DTR> PRINT TEXT_STRING OF ADT_TEXT WITH
CON> ACCESS_KEY = THIS.ACCESS_KEY
```

```
TEXT
STRING
```

DATATRIEVE creates a file with the name you specify.

For example, if you enter EMPLOYEE,

DATATRIEVE creates a file named EMPLOYEE.DAT.

DATATRIEVE uses this file to store your data.

DTR>

This example shows that you can add records to DTRADT.DAT. The records you add should use an existing ACCESS_KEY. If you define a record with a new ACCESS_KEY, ADT does not display the associated text.

Customizing ADT and Guide Mode

7.1 Customizing ADT

7.1.2.4 Messages During an ADT Session

Some text that you see during an ADT session is not in the DTRADT.DAT file. For example, if you incorrectly answer a prompt that requires a YES or NO, you get the following message:

```
Please answer with either YES (Y) or NO (N).
```

This message and a number of others are located in the DATATRIEVE message file DTRMSG.SYS. Messages specific to ADT begin with the letters ADT. If you want to modify these messages, follow the procedure described in Chapter 6.

7.1.3 Replace the Old ADT Text File with the New One

If you want ADT to display your new text to all DATATRIEVE users, copy your new DTRADT.DAT file to SYS\$LIBRARY:

```
$ COPY DTRADT.DAT SYS$LIBRARY:DTRADT.DAT/PROTECTION=WORLD:R
```

You must have write access to SYS\$LIBRARY to do this.

If you want DATATRIEVE to display the new ADT text only to selected users, assign the process logical name DTRADT to the new DTRADT.DAT file for each of these users. For example, if the new DTRADT.DAT file is in the DB0:[CASSIDY.DTR] directory, use the following command:

```
$ ASSIGN "DB0:[CASSIDY.DTR]DTRADT.DAT" DTRADT
```

Users who want DATATRIEVE to display the new ADT text can include this command in their LOGIN.COM files.

DATATRIEVE supplies the file specification SYS\$LIBRARY:DTRADT.DAT, which becomes SYS\$SYSROOT:[SYSLIB]DTRADT.DAT. If the file specification in the ASSIGN command is not complete, DATATRIEVE does not replace that part of the specification. For example:

```
$ ASSIGN "DBA3:DTRADT" DTRADT
```

The full file specification for the ADT text file becomes DBA3:[SYSLIB]DTRADT.DAT. DATATRIEVE will not look in your default directory for the file.

7.2 Customizing Guide Mode

You can invoke Guide Mode at two different levels: simple and advanced. When you enter the command SET GUIDE, you get the simple level of Guide Mode. If you enter SET GUIDE ADVANCED, you get the advanced level.

Customizing ADT and Guide Mode

7.2 Customizing Guide Mode

The difference between these levels is the commands, statements, or options you can use. Table 7–1 shows what is available at the simple and advanced levels.

Table 7–1 Guide Mode Levels

Command, Statement, or Option	Level at which it is available
FIND	Both
MODIFY	Both
PLOT	Advanced
PRINT	Both
READY	Both
REPORT	Advanced
SELECT	Both
SET	Both
SHOW	Both
SORT	Both
STORE	Both
: (to invoke procedures)	Advanced
AUTO_FINISH	Neither

This table shows the default distribution of the available commands and statements. You can change the default and specify which commands are available at what level. You can also use the AUTO_FINISH option to specify whether or not DATATRIEVE completes words as they are entered at the terminal.

The following steps are necessary to customize Guide Mode:

1. Copy the DTRGUIDE.DAT file to one of your own directories.
2. Use DATATRIEVE to modify the distribution of commands, statements, and options available at each Guide Mode level.
3. Replace the old DTRGUIDE.DAT file with the new one.

The following sections describe each of these steps.

In addition to distributing features between GUIDE and GUIDE ADVANCED, you can customize all of the Guide Mode prompts and informational text. This text is located in DTR\$LIBRARY:DTRMSG.SMSG. Prompt messages begin with

Customizing ADT and Guide Mode

7.2 Customizing Guide Mode

GPR and informational messages begin with GUI. Chapter 6 explains how to customize these messages.

7.2.1 Copy the DTRGUIDE.DAT File

The file DTRGUIDE.DAT is located in SYS\$LIBRARY. It contains data that defines the distribution of commands, statements, and options in simple and advanced Guide Mode.

You can copy DTRGUIDE.DAT to one of your directories as follows:

```
$ COPY SYS$LIBRARY:DTRGUIDE.DAT DB0:[HAMMOND.DTR]
```

You can also have DATATRIEVE create a new DTRGUIDE.DAT file as follows:

```
DTR> SET DICTIONARY CDD$TOP.DTR$LIB.GUIDE
DTR> :STORE_DEFAULT_OPTIONS
```

7.2.2 Modify the Distribution of Commands and Statements

After you have a copy of DTRGUIDE.DAT in your default VMS directory, invoke DATATRIEVE and set your default dictionary directory to CDD\$TOP.DTR\$LIB.GUIDE. You can then invoke the procedure LIST_OPTIONS to see the current distribution:

```
DTR> SET DICTIONARY CDD$TOP.DTR$LIB.GUIDE
DTR> :LIST_OPTIONS
STORE_OPT      : BOTH
MODIFY_OPT     : BOTH
PLOT_OPT       : ADVANCED
REPORT_OPT     : ADVANCED
PRINT_OPT      : BOTH
SORT_OPT       : BOTH
SELECT_OPT     : BOTH
FIND_OPT       : BOTH
READY_OPT      : BOTH
SHOW_OPT       : BOTH
SET_OPT        : BOTH
PROCEDURE_OPT  : ADVANCED
AUTO_FINISH    : NEITHER

DTR>
```

The left-hand column lists each command, statement, or option available in Guide Mode. The right-hand column indicates at what level each is available. You have the following choices for each command, statement, or option:

- **BOTH (0)**
The command, statement, or option can be used in both levels of Guide Mode.
- **ADVANCED (1)**

Customizing ADT and Guide Mode

7.2 Customizing Guide Mode

The command, statement, or option can be used only in advanced Guide Mode.

- SIMPLE (2)

The command, statement, or option can be used only in simple Guide Mode.

- NEITHER (3)

The command, statement, or option cannot be used in either Guide Mode.

To see what choice corresponds to what number, enter:

```
DTR> SHOW OPTIONS_TABLE
```

To change the distribution of commands, statements, and options, use the domain OPTIONS. Ready the domain for modify access and print its single record:

```
DTR> READY OPTIONS MODIFY
DTR> PRINT OPTIONS
```

STORE	MODIFY	PLOT	REPORT	PRINT	SORT	SELECT	FIND	READY	SHOW	SET	PROCEDURE	AUTO
OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	OPT	FINISH
0	0	1	1	0	0	0	0	0	0	0	1	3

```
DTR>
```

Change the distribution of commands, statements, or options to suit your needs.

For example:

```
DTR> FIND OPTIONS
[1 record found]
DTR> SELECT
DTR> !Make the plot statement available at both levels
DTR> MODIFY PLOT_OPT
Enter PLOT_OPT: 0
DTR> !Make the STORE statement available only in advanced Guide Mode
DTR> MODIFY USING STORE_OPT = "ADVANCED" VIA OPTIONS_TABLE
DTR>
```

7.2.3 Replace the Old DTRGUIDE.DAT with the New One

If you want your new arrangement of simple and advanced Guide Mode to be the default for all users, copy the new DTRGUIDE.DAT file to SYS\$LIBRARY:

```
$ COPY DTRGUIDE.DAT SYS$LIBRARY:DTRGUIDE.DAT/PROTECTION=WORLD:R
```

You must have write access to SYS\$LIBRARY to do this.

If you want your arrangement to be available only to selected users, assign the process logical name DTRGUIDE to the new DTRGUIDE.DAT file for those users. For example, if the new DTRGUIDE.DAT is in the directory DB0:[HAMMOND.DTR], use the following command:

```
$ ASSIGN "DB0:[HAMMOND.DTR]DTRGUIDE.DAT" DTRGUIDE
```

Customizing ADT and Guide Mode

7.2 Customizing Guide Mode

Users who want to use your arrangement of Guide Mode can include this command in their LOGIN.COM file.

Note that DATATRIEVE supplies the default file specification of SYS\$LIBRARY:DTRGUIDE.DAT, which becomes SYS\$SYSROOT:[SYSLIB]DTRGUIDE.DAT. If the file specification in the ASSIGN command is not complete, DATATRIEVE does not replace that part of the specification. For example:

```
$ ASSIGN "DBA3:DTRGUIDE" DTRGUIDE
```

The full file specification for the Guide Mode file becomes DBA3:[SYSLIB]DTRGUIDE.DAT. DATATRIEVE will not look in your default directory for the file.

8

Customizing DATATRIEVE Text

Chapters 5, 6, and 7 explain how to customize DATATRIEVE help text, messages, ADT, and Guide Mode. There are other text elements that DATATRIEVE understands or displays. This chapter explains how to customize these additional text elements.

To customize DATATRIEVE text, you must create and install a new DATATRIEVE shareable image. This requires the privileges SYSPRV and CMKRNL.

8.1 DATATRIEVE Text

You can modify several types of DATATRIEVE text:

- Syntax prompts
- Responses to SHOW commands
- Date text
- Default edit strings
- Statistical expressions
- Keywords

The following sections give examples of each type of text.

8.1.1 Syntax Prompt Text

Some DATATRIEVE text is embedded in the DATATRIEVE syntax prompt message, which is defined in the DATATRIEVE message file DTRMSG.SMSG. Here is an example:

Customizing DATATRIEVE Text

8.1 DATATRIEVE Text

```
DTR> READY
[Looking for dictionary path name]      1
CON> PERSONNEL
DTR> FIND
[Looking for name of domain, collection, or list]  1
CON> PERSONNEL
[24 records found]
DTR> SORT
[Looking for sort list]      1
CON> BY LAST_NAME
DTR>
```

Each of the messages highlighted by the callout **1** is an instance of `DTR$_LOOKINFO`, the DATATRIEVE syntax prompt. The `DTR$_LOOKINFO` message has the following format:

```
[Looking for <text string>]
```

The text string DATATRIEVE inserts in this message depends on the context of your command or statement.

If you want to change the “Looking for” part of the message, follow the procedure described in Chapter 6. The remainder of this chapter explains how to change DATATRIEVE text such as “dictionary path name,” “name of domain, collection, or list,” and “sort list.”

8.1.2 SHOW Text

Another kind of text is what DATATRIEVE displays in response to a `SHOW` command. Here is an example:

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

In this example, you can change the text that describes the fields Primary key, Number, Character string, and Date.

Customizing DATATRIEVE Text

8.1 DATATRIEVE Text

8.1.3 Date Text

DATATRIEVE text that you can customize includes date text, as illustrated in the following example:

```
DTR> DECLARE X DATE.  
DTR> X = "YESTERDAY"  
DTR> PRINT X
```

```
      X  
16-Sep-1982
```

```
DTR>
```

In this example, you can change the DATATRIEVE date value expression “YESTERDAY.” You can also change the value expressions “TODAY,” “TOMORROW,” and “NOW.” For example, you can translate these words or month and day names into a foreign language.

8.1.4 Default Edit Strings

In the previous example, DATATRIEVE displays a date in the default format:

```
16-Sep-1982
```

You can change the default edit strings for dates, standard deviations, and floating point numbers.

8.1.5 Statistical Text

You can use a number of statistical expressions such as COUNT, TOTAL, and AVERAGE to summarize your data. Here is an example:

```
DTR> FIND PERSONNEL WITH DEPT = "D98"  
[5 records found]  
DTR> PRINT COUNT, AVERAGE SALARY, STD_DEV SALARY
```

```
      STANDARD  
      AVERAGE DEVIATION  
COUNT SALARY   SALARY  
      5 $40,095  1.3369E+04
```

```
DTR>
```

In this example, you can change the default headers COUNT, AVERAGE and STANDARD DEVIATION.

Customizing DATATRIEVE Text

8.1 DATATRIEVE Text

8.1.6 Keywords

You can customize all DATATRIEVE keywords by using the DECLARE SYNONYM statement.

You can also customize the following keywords as DATATRIEVE text:

ADT	HELP_WINDOW
CLOSE	HELP_LINES
EDIT	NO_HELP_PROMPT
EXIT	NO_HELP_WINDOW
GUIDE	OPEN
HELP	@ (command file invocation)
HELP_PROMPT	

If you change any of these commands in the text file, DATATRIEVE no longer recognizes the original command as a keyword. For example, if you translate HELP to HILFE, you remove the keyword HELP from the DATATRIEVE vocabulary. If you need to use both the HELP and HILFE commands, you can declare HELP as a synonym for HILFE.

8.2 How to Change DATATRIEVE Text

Follow these steps to change DATATRIEVE text:

1. Copy the DATATRIEVE text file to one of your own directories.
2. Edit the text file to make the desired changes.
3. Assemble the text file to produce the text object file.
4. Replace the text object file in DTRLIBxx.OLB.
5. Relink the DATATRIEVE shareable image.

The following sections describe each of these steps.

8.2.1 Copy the DATATRIEVE Text File

The DATATRIEVE text file DTRTEXT.MAR is located in DTR\$LIBRARY. Use the COPY command to copy this file to one of your own directories:

```
$ COPY DTR$LIBRARY:DTRTEXT.MAR DB0:[NAPRA]*.*
```

This command creates a copy of DTRTEXT.MAR in the DB0:[NAPRA] directory.

Customizing DATATRIEVE Text 8.2 How to Change DATATRIEVE Text

8.2.2 Edit the Text File

The DATATRIEVE text file DTRTEXT.MAR is written in VAX MACRO. It contains all the DATATRIEVE text entries. Here is one of them:

```
$DTR$TEXT_ENTRY DTR$$K_TXT_PRM_RSE,-  
                ^\name of domain, collection, or list\  
                \
```

This entry defines a prompting text string. DATATRIEVE displays this text in its “[Looking for . . .]” syntax prompt.

Each entry in DTRTEXT.MAR has a similar format. The DATATRIEVE text is enclosed by the opening circumflex and backslash (^ \) and closing backslash (\) delimiters. This is the only part of the text file you should change. Changing any other part of the file may cause errors when you attempt to assemble DTRTEXT.MAR or in the text DATATRIEVE displays.

Use an editor to make the desired changes to the text within the ^\ and \ delimiters. For example, if you want to change the default format that DATATRIEVE uses to display dates, find the following entry:

```
$DTR$TEXT_ENTRY DTR$$K_TXT_DAT_PIC,-  
                ^\DD-MMM-YYYY\  
                \
```

Change the text to:

```
                ^\DD-M(9)-YYYY\  
                \
```

Here is the result of this change:

```
DTR> DECLARE X DATE.  
DTR> X = "YESTERDAY"  
DTR> PRINT X
```

X

16-September-1987

DTR>

If you want to change the value expression “YESTERDAY,” find the entry in DTRTEXT.MAR:

```
$DTR$TEXT_ENTRY DTR$$K_TXT_DAT_YESTER,-  
                ^\YESTERDAY\  
                \
```

Now use an editor to change the string “YESTERDAY.” This is all that is required to edit the DATATRIEVE text file.

Customizing DATATRIEVE Text

8.2 How to Change DATATRIEVE Text

8.2.3 Assemble the Text File

After you edit DTRTEXT.MAR, assemble it with the following command:

```
$ MACRO DTRTEXT
```

This command creates the object file DTRTEXT.OBJ in your default directory.

8.2.4 Replace the Text Object File in DTRLIBxx.OLB

After you create the text object file, replace it in the library file DTR\$LIBRARY:DTRLIBxx.OLB. If the new object file is in your default directory, you can use the command:

```
$ LIBRARY/REPLACE DTR$LIBRARY:DTRLIBxx DTRTEXT
```

8.2.5 Relink the DATATRIEVE Shareable Image

After you replace the DTRTEXT.OBJ in the library file, you must relink the DATATRIEVE shareable image. Use the command procedure DTRBLDxx.COM to relink DATATRIEVE. When you relink, you must be sure that the shareable image you relink against is from the most recently installed version of DATATRIEVE.

To relink VAX DATATRIEVE, execute the command file DTRBLDxx:

```
$ @DTR$LIBRARY:DTRBLDxx
```

9

Translating DATATRIEVE

This chapter shows you how to translate DATATRIEVE into a foreign language. It describes how you should plan your translation and how to implement the translation.

You can translate all DATATRIEVE information management services for speakers of foreign languages. For example, here is a sample dialogue with DATATRIEVE in German:

```
$ DTR32
VAX DATATRIEVE V5.0
DEC Abfrage und Report System
Tippe HILFE fuer Hilfe
DTR> ZEIGE BEREICHE
Bereiche:
    * BESITZER      * FAMILIEN      * JACHTEN      * JAHRESBERICHT
    * PERSONAL

DTR> AKTIVIERE PERSONAL
DTR> ZEIGE BERIRT
Element "BERIRT" nicht im Lexikon gefunden.
DTR> ZEIGE ACTIVIIERT
Bereitete Bereiche:
    PERSONAL: Bereich, RMS indiziert, geschuetzter Lesezugriff
               <CDD$TOP.KELLER.PERSONAL>
Keine Tabellen geladen.

DTR> ZEIGE FELDER FUER PERSONAL
PERSONAL
PERSON
    AUSWEISNUMMER (NUMMER)    <Zahl, indiziertes Schluesselfeld>
    STATUS    <Zeichenfolge>
    NAME (NAME)
        VORNAME    <Zeichenfolge>
        NACHNAME    <Zeichenfolge>
    ABTEILUNG (ABT)    <Zeichenfolge>
    ANFANGSDATUM (DATUM)    <Datum>
    GEHALT    <Nummer>
    VORGESETZTER_NUMMER    <Nummer>
```

Translating DATATRIEVE

```
DTR> SUCHE PERSON IN PERSONAL MIT ABTEILUNG = "D98"  
[4 Sätze gefunden]  
DTR> DRUCKE NACHNAME, NUMMER, DATUM, GEHALT VON  
[Suche Name eines Bereichs, einer Sammlung, oder Liste]  
CON> PERSON SORTIERT NACH DATUM DRUCKE  
[Suche Anweisung]  
CON> DRUCKE SPALTE 30, SUMME GEHALT ("GESAMTGEHALT") IM AUSGABEFORMAT  
[Suche Masken- oder Aufbereitungszeichenkette]  
CON> 999B999B"DM"
```

NACHNAME	AUSWEIS NUMMER	ANFANGS DATUM	GEHALT
NALEVO	84375	3-Jan-1976	56 847 DM
TERRICK	39485	2-Mai-1977	55 829 DM
TERRY	02943	2-Jan-1980	29 908 DM
HAMMER	49843	4-Aug-1981	26 392 DM
		GESAMTGEHALT	
			168 976 DM

```
DTR> ENDE  
$
```

As the example shows, you can make DATATRIEVE understand and respond in a foreign language. This chapter explains how.

9.1 Planning Your Translation

Before you begin your translation, you should read Chapters 4, 5, 6, 7, and 8 to learn about the elements of DATATRIEVE you need to translate. Knowing what these elements are and how to customize them will help you plan your translation.

DATATRIEVE runs on the VMS operating system and works with a number of other software products. In designing your translation of DATATRIEVE, you should take into account how these interrelated products will affect your DATATRIEVE translation. For example, when you invoke a text editor such as EDT or VAXTPU with the EDIT command, the messages and help text that the editor displays belong to the editor and not to DATATRIEVE. Thus, to completely translate DATATRIEVE, you also need to translate the help text and messages for the editor, if possible.

Another service DATATRIEVE uses is the VMS Help facility. The help text DATATRIEVE displays is part of DATATRIEVE. However, the prompts "Topic?" and "Subtopic?" at the bottom of the display come from the VMS Help facility. To translate these prompts, you need to translate the VMS Help library.

Translating DATATRIEVE

9.1 Planning Your Translation

Other text DATATRIEVE displays come from a VAX CDD/Repository dictionary, VAX Record Management Services (RMS), VAX DBMS, and other VAX software products. This chapter discusses only the translation of DATATRIEVE.

9.2 How to Translate DATATRIEVE

To translate DATATRIEVE, you need to translate:

- Keywords
- Help text
- Messages
- ADT text
- Remaining text elements
- Names of functions

Then you need to relink the DATATRIEVE shareable image. To create a new shared image you must have the privileges SYSPRV and CMKRNL.

9.2.1 Translating Keywords

In order to make DATATRIEVE respond to commands and statements in a foreign language, you must translate all DATATRIEVE keywords. Refer to the *VAX DATATRIEVE Reference Manual* for a list of keywords.

You can translate keywords by declaring foreign-language synonyms for each keyword and inserting these declarations in your DATATRIEVE startup command file.

If you do not have a DATATRIEVE startup command file, create one as follows:

1. Use an editor to create a file. For example:

```
$ EDIT DB2:[KELLER.DTR]DTRSTART.COM
```

2. Assign the logical name DTR\$STARTUP to this file. For example, insert the following command in your LOGIN.COM file:

```
$ ASSIGN "DB2:[KELLER.DTR]DTRSTART.COM" DTR$STARTUP
```

Each time you invoke DATATRIEVE, DATATRIEVE executes the contents of your startup command file. Thus, you can include your keyword translations in this file. For example, you can insert the following statement into your startup command file:

Translating DATATRIEVE

9.2 How to Translate DATATRIEVE

```
DECLARE SYNONYM ALLES    FOR ALL,  
              ALLE      FOR ALL,  
              BEREICH   FOR DOMAIN,  
              BEREICHE FOR DOMAINS,  
              AKTIVIERE FOR READY,  
              ACTIVIERT FOR READY,  
              DATUM     FOR DATE,  
              DRUCKE    FOR PRINT,  
              HILFE     FOR HELP,  
              ZEIGE     FOR SHOW
```

Each phrase in this statement creates a German equivalent for an English keyword.

Note that some English keywords require more than one German synonym. For example, the English word READY has a different function in the following examples:

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

In German, each of these uses of READY requires a different translation:

```
DTR> AKTIVIERE PERSONNEL  
DTR> ZEIGE ACTIVIERT
```

Other DATATRIEVE keywords require multiple translations. For example, the keyword ALL requires two translations in German and three in Spanish:

<i>English</i>	<i>German</i>	<i>Spanish</i>
SHOW ALL	ZEIGE ALLES	MUESTRA TODO
PRINT ALL EMPLOYEES	DRUCKE ALLE ANGESTELLTE	IMPRIME TODOS EMPLEADOS IMPRIME TODAS EMPLEADAS

9.2.2 Translating Help Text

After you translate the DATATRIEVE keywords, you can translate help text for them. Use the following procedure:

1. Copy the DATATRIEVE help library to one of your directories:

```
$ COPY SYS$HELP:DTRHELP.HLB  
$_To:          DB2:[KELLER.HELP]NEWHELP.HLB
```

2. Use the LIBRARY command to extract all the help text from the library:

```
$ LIBRARY/HELP/EXTRACT=* NEWHELP
```

This command creates the text file NEWHELP.HLP in your default directory.

Translating DATATRIEVE

9.2 How to Translate DATATRIEVE

3. Translate the text file NEWHELP.HLP. The following example shows the help text for the statement FIND:

1 FIND

The FIND statement brings together a collection of records from a domain or a previously established collection. The form of the statement is:

```
FIND rse
```

where rse is a record selection expression (see HELP RSE). The number of records found is printed on your terminal.

For example, gather a collection of yachts:

```
FIND YACHTS WITH LENGTH_OVER_ALL BETWEEN 26 AND 30
```

This makes a collection named CHEAP_ONES:

```
FIND CHEAP_ONES IN YACHTS WITH PRICE < 15000
```

Use a text editor to translate this text. The following example shows a sample translation:

1 SUCHE

Die SUCHE Anweisung enthaelt eine Satzsammlung von einem Bereich oder von einer vorher aufgestellten Sammlung. Das Format der Anweisung ist:

```
SUCHE sa
```

worin "sa" ein Auswahlkriterium ist (siehe HILFE SA). Die Zahl der gefundenen Saetze ist auf Ihrem Bildschirm angezeigt.

Zum Beispiel, stelle eine Sammlung von JACHTEN auf:

```
SUCHE JACHTEN MIT LAENGE ZWISCHEN 26 UND 30
```

Oder, stelle eine Sammlung genannt BILLIGE auf:

```
SUCHE BILLIGE IN JACHTEN MIT PREIS < 15000
```

4. Replace the translated text file NEWHELP.HLP in the help library:

```
$ LIBRARY/HELP/REPLACE
_Library:  NEWHELP.HLB
_File:    NEWHELP.HLP
```

5. Replace the old DTRHELP.HLB file with the new one. You can do this in two ways:

- To access the translated text from your current process, assign the logical name DTRHELP to NEWHELP.HLB:

```
$ ASSIGN "DB2:[KELLER.HELP]NEWHELP.HLB" DTRHELP
```

Translating DATATRIEVE

9.2 How to Translate DATATRIEVE

- To update the help file for all users on the system, copy the NEWHELP.HLB file to DTRHELP.HLB in the SYS\$HELP directory (you must have write access to SYS\$HELP to do this):

```
$ COPY DB2:[KELLER.HELP]NEWHELP.HLB
_To:          SYS$HELP:DTRHELP.HLB
```

When you invoke DATATRIEVE and enter HILFE SUCHE, DATATRIEVE displays your translated help text.

For more detailed information about modifying DATATRIEVE help text, refer to Chapter 5.

9.2.3 Translating Messages

To translate DATATRIEVE messages follow these steps:

1. Copy the DATATRIEVE message source file to one of your own directories:

```
$ COPY DTR$LIBRARY:DTRMSG.SMSG
_To:          DB2:[KELLER.MSG]NEWMSG.SMSG
```

This command creates a copy of DTRMSG.SMSG called NEWMSG.SMSG in your default directory.

2. Edit the message source file to make the desired changes. For example, to change the top line of the ADT video display, find the following message definition:

```
ADTIVIDEO/FAO=0/INFORMATIONAL-
<A D T ? - Help ! - Fields < - Back up PF2 - Screen Help>
```

Translate the text portion of the message definition:

```
ADTIVIDEO/FAO=0/INFORMATIONAL-
<A D T ? - Hilfe ! - Felder < - Zurueckgehen PF2 - Bildschirm Hilfe>
```

3. Compile the translated message source file to create an object file:

```
$ MESSAGE DB2:[KELLER.MSG]NEWMSG.SMSG
```

4. Link the object file to create the file NEWMSG.EXE:

```
$ LINK/EXECUTABLE=NEWMSG.EXE NEWMSG.OBJ
```

5. Replace the old DTRMSG.EXE file with the new one. You can do this in two ways:

- To access the translated text from your current process, assign the logical name DTRMSG to NEWMSG.EXE:

```
$ ASSIGN "DB2:[KELLER.MSG]NEWMSG.EXE" DTRMSG
```


Translating DATATRIEVE

9.2 How to Translate DATATRIEVE

- To update the message file for all users on the system, copy the NEWMSG.S.EXE to DTRMSG.S.EXE in the SYS\$MESSAGE directory. (You must have write access to SYS\$MESSAGE to do this.)

```
$ COPY DB2:[KELLER.MSG]NEWMSG.S.EXE
_To:          SYS$MESSAGE:DTRMSG.S.EXE
```

For more detailed information about modifying DATATRIEVE messages, refer to Chapter 6.

9.2.4 Translating ADT

To translate ADT, follow these steps:

1. Copy the ADT text file to one of your own directories:

```
$ COPY SYS$LIBRARY:DTRADT.DAT
_To:          DB0:[CASS.DTR]DTRADT.DAT
```

2. Invoke DATATRIEVE and use the ADT_TEXT domain to translate text entries:

```
DTR> SET DICTIONARY CDD$TOP.DTR$LIB.ADT
DTR> READY ADT_TEXT WRITE
DTR> FIND ADT_TEXT WITH TEXT_STRING CONT
CON> "Enter domain name : "
[1 record found]
DTR> MODIFY CURRENT USING
CON> TEXT_STRING = "Bereichsname eingeben : "
DTR>
```

3. Replace the old DTRADT.DAT file with the new one. You can do this in two ways:

- To access the translated text from your current process, assign the logical name DTRADT to your version of the file:

```
$ ASSIGN "DB0:[CASS.DTR]DTRADT" DTRADT
```

- To update the text file for all users on the system, copy the new DTRADT.DAT to the SYS\$LIBRARY directory. (You must have write access to SYS\$LIBRARY to do this.)

```
$ COPY DB0:[CASS.DTR]DTRADT.DAT
_To:          SYS$LIBRARY:DTRADT.DAT
```

For more detailed information about modifying ADT text, refer to Chapter 7.

Translating DATATRIEVE

9.2 How to Translate DATATRIEVE

9.2.5 Translating the Remaining Text Elements

After you translate help, ADT, and messages, you need to translate the remaining DATATRIEVE display text. Use the following procedure to translate this text:

1. Copy the DATATRIEVE text file to one of your own directories:

```
$ COPY DTR$LIBRARY:DTRTEXT.MAR
_To:          DB2:[KELLER.TXT]DTRTEXT.MAR
```

2. Edit the text file to make the desired changes. For example, to translate the month names DATATRIEVE displays, find the text entries for each month:

```
$DTR$TEXT_ENTRY DTR$$K_TXT_DAT_JAN, -
                ^\January\
.
.
.
$DTR$TEXT_ENTRY DTR$$K_TXT_DAT_DEC, -
                ^\December\
```

Translate the text between the backslashes:

```
$DTR$TEXT_ENTRY DTR$$K_TXT_DAT_JAN, -
                ^\Januar\
.
.
.
$DTR$TEXT_ENTRY DTR$$K_TXT_DAT_DEC, -
                ^\Dezember\
```

3. When you finish translating the text entries, assemble the text file to produce the text object file:

```
$ MACRO DB2:[KELLER.TXT]DTRTEXT
```

This command creates the file DTRTEXT.OBJ in your default directory.

4. Replace the text object file in the object library DTRLIBxx.OLB:

```
$ LIBRARY/REPLACE DTR$LIBRARY:DTRLIBxx.OLB
_File:          DB2:[KELLER.TXT]DTRTEXT.OBJ
```

To see the results of your translation, you must relink DATATRIEVE as described at the end of this chapter.

For more detailed information about modifying DATATRIEVE text, refer to Chapter 8.

Translating DATATRIEVE 9.2 How to Translate DATATRIEVE

9.2.6 Translating the Names of Functions

You can use a number of DATATRIEVE functions to perform specific tasks. The *VAX DATATRIEVE Reference Manual* describes these functions. Here is one example:

```
DTR> PRINT FN$$SQRT(2)
      FN$$SQRT
      1.4142E+00
DTR>
```

The function in this example is FN\$\$SQRT. You can translate the name of this and all other DATATRIEVE functions. You can also translate the default header that DATATRIEVE displays when you invoke the function. Thus, you can invoke the FN\$\$SQRT as follows:

```
DTR> DRUCKE QUADRATWURZEL(2)
      Quadratwurzel
      1.4142E+00
DTR>
```

To translate functions, follow these steps:

1. Copy the function definition file to one of your own directories:

```
$ COPY DTR$LIBRARY:DTRFNDxx.MAR
   _To:      DB2:[KELLER.FUN]DTRFNDxx.MAR
```

2. Edit the function definition file. For example, here is the definition of the function FN\$\$SQRT:

```
$DTR$FUN_DEF  FN$$SQRT, MTH$$SQRT, 1
      $DTR$FUN_OUT_ARG  TYPE = FUN$K_VALUE, DTYPE = DSC$K_DTYPE_F
      $DTR$FUN_IN_ARG   TYPE = FUN$K_REF, DTYPE = DSC$K_DTYPE_F, -
      ORDER = 1
$DTR$FUN_END_DEF
```

Change the name of the function in the \$DTR\$FUN_DEF statement. For example, change the name FN\$\$SQRT to QUADRATWURZEL, the word for square root in German. Next, use the \$DTR\$FUN_HEADER statement to specify the default header for the function value. For example, insert the following statement after DTR\$FUN_DEF:

```
$DTR$FUN_HEADER      HDR = <"Quadratwurzel">
```

Translating DATATRIEVE

9.2 How to Translate DATATRIEVE

Your translated function definition is as follows:

```
$DTR$FUN_DEF    QUADRATWURZEL, MTH$SQRT, 1
  $DTR$FUN_HEADER  HDR = <"Quadratwurzel">
  $DTR$FUN_OUT_ARG TYPE = FUN$K_VALUE, DTYPE = DSC$K_DTYPE_F
  $DTR$FUN_IN_ARG  TYPE = FUN$K_REF, DTYPE = DSC$K_DTYPE_F, -
                    ORDER = 1
$DTR$FUN_END_DEF
```

3. After you finish editing DTRFNDxx.MAR, assemble it with the command:

```
$ MACRO DB2:[KELLER.FUN]DTRFNDxx.MAR
```

This command creates the object file DTRFNDxx.OBJ.

4. Replace DTRFNDxx.OBJ in the function library DTR\$LIBRARY:DTRFUNxx. Use the following command:

```
$ LIBRARY/REPLACE DTR$LIBRARY:DTRFUNxx.OLB
  _File:           DB2:[KELLER.FUN]DTRFNDxx.OBJ
```

You should now relink the DATATRIEVE shareable image.

9.2.7 Relinking the DATATRIEVE Shareable Image

After you replace DTRFNDxx.OBJ in the object module library, you must relink the DATATRIEVE shareable image. You use the command procedure DTRBLDxx.COM to relink. When you relink, you must be sure that the shareable image you link against is from the most recently installed version of DATATRIEVE.

To relink DATATRIEVE, execute the command file DTR\$LIBRARY:DTRBLDxx.COM:

```
$ @DTR$LIBRARY:DTRBLDxx
```

Note that the files DTRFNDxx.MAR, DTRFUNxx.OLB, DTRLIBxx.OLB, and DTRBLDxx.COM may have a suffix appended to the file name. For example, DTRBLDxx.COM might be DTRBLDFMS.COM.

A

Definitions of the DATATRIEVE Access Block

This appendix points you to directory locations where you can find definitions of the DATATRIEVE Access Blocks (DABs). It also tells you where to find sample programs and, finally, it describes the structure of the DAB for users who need to define the DAB themselves to call DATATRIEVE routines from other VAX languages.

A.1 Location of DATATRIEVE Access Blocks

The definitions of the DATATRIEVE Access Block (DAB) in VAX FORTRAN, VAX COBOL, VAX BASIC, VAX Pascal, VAX PL/I, and VAX C are available on-line. Copies of these inclusion files are supplied with the DATATRIEVE installation kit and are placed in the DTR\$LIBRARY directory.

Table A-1 lists the names of the DAB files for each language.

Table A-1 DAB Definition Files in DTR\$LIBRARY

Language	File Name
VAX BASIC	DTR\$LIBRARY:DAB.BAS
VAX COBOL	DTR\$LIBRARY:DAB.LIB
VAX FORTRAN	DTR\$LIBRARY:DAB.FOR
VAX Pascal	DTR\$LIBRARY:DAB.PAS
VAX PL/I	DTR\$LIBRARY:DAB.PLI
VAX C	DTR\$LIBRARY:DAB.H

Definitions of the DATATRIEVE Access Block

A.2 Location of Sample Programs

A.2 Location of Sample Programs

Users unfamiliar with DATATRIEVE can use the following FORTRAN, COBOL, BASIC, Pascal, PL/I, and C programs to display data from three sample DATATRIEVE domains. These programs are located in the DTR\$LIBRARY directory. Table A-2 shows you the programs name and the programming language they refer to.

Table A-2 Sample Programs

Programming Language	Program Name
FORTRAN	EXAMPLE.FOR
COBOL	EXAMPLE.COB
BASIC	EXAMPLE.BAS
Pascal	EXAMPLE.PAS
PL/I	EXAMPLE.PLI
C	EXAMPLE.C

To run these programs, you need copies of the sample data files and domain definitions that the DATATRIEVE installation procedure provides. See the *VAX DATATRIEVE User's Guide* for information on running the NEWUSER.COM procedure to set up the sample databases.

Furthermore, you can find sample FORTRAN, BASIC, and COBOL programs that call DATATRIEVE in the DTR\$LIBRARY directory. These programs show you how you can call DATATRIEVE to perform calculations on data, store and retrieve data, and create data management applications for end users.

Table A-3 lists the names of the program files for each language.

Table A-3 Files Containing Sample Programs that Call DATATRIEVE

Language	File Name
VAX FORTRAN	CALCULATE.FOR, CHOOSE.FOR, CLEAR.FOR, COMMON.FOR, CORRELATE.FOR, DISPLAY.FOR, ESTABLISH.FOR, MENU.FOR, MESSAGE.FOR, MODIFY.FOR, PARSE.FOR, REPORT.FOR, SORT.FOR, STORE.FOR
VAX BASIC	COLUMNS.BAS, LINEAR.BAS
VAX COBOL	ENTRY.COB, PAYROLL.COB, UPDATE.COB

Definitions of the DATATRIEVE Access Block

A.3 Defining the DATATRIEVE Access Block in Other VAX Languages

A.3 Defining the DATATRIEVE Access Block in Other VAX Languages

You can use the DATATRIEVE Call Interface with any VAX high-level language that complies with the VAX Calling Standard. However, if you use a language other than FORTRAN, COBOL, BASIC, Pascal, PL/I, or C, you will have to define the DATATRIEVE Access Block within your program.

To create your own DAB, you must define the following:

- The access block
- The message buffers
- DATATRIEVE constants

The following sections describe each component of the DATATRIEVE Access Block separately.

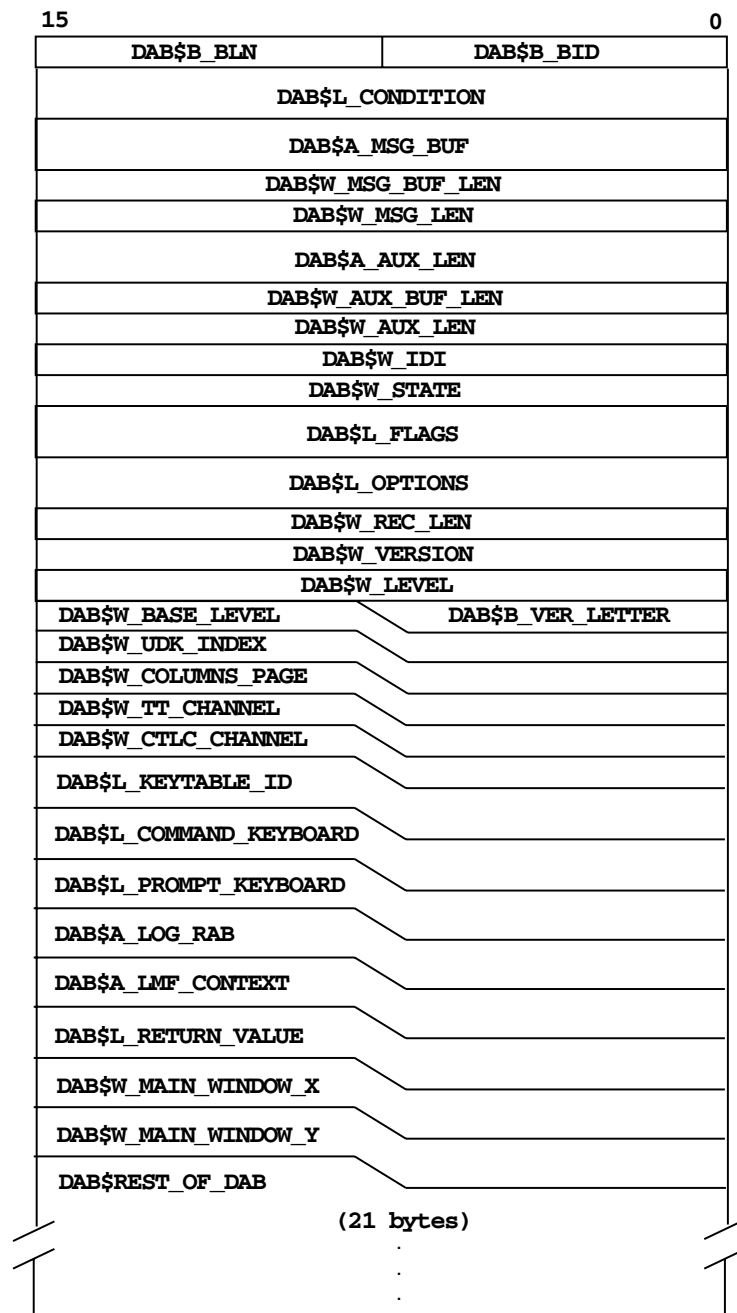
A.3.1 Defining the Access Block

The DATATRIEVE Access Block is a 100-byte record block that DATATRIEVE and your program use to communicate. Figure A-1 illustrates the structure of the Access Block and Table A-4 describes the data type and purpose of each field.

Definitions of the DATATRIEVE Access Block

A.3 Defining the DATATRIEVE Access Block in Other VAX Languages

Figure A-1 Structure of the DATATRIEVE Access Block



Definitions of the DATATRIEVE Access Block

A.3 Defining the DATATRIEVE Access Block in Other VAX Languages

Table A-4 Fields of the DATATRIEVE Access Block

DAB Field Name	Data Type	Content of Field
DAB\$B_BID	Byte integer	Reserved to Digital
DAB\$B_BLN	Byte integer	Reserved to Digital
DAB\$L_CONDITION	Longword integer	A condition code that identifies the status of the last call to DATATRIEVE
DAB\$A_MSG_BUF	Longword integer, unsigned	Address of the message buffer
DAB\$W_MSG_BUF_LEN	Word integer	Length of the message buffer
DAB\$W_MSG_LEN	Word integer	Length of the current message string stored in the message buffer
DAB\$A_AUX_BUF	Longword integer, unsigned	Address of the auxiliary message buffer
DAB\$W_AUX_BUF_LEN	Word integer	Length of the auxiliary message buffer
DAB\$W_AUX_LEN	Word integer	Length of the current message string stored in the auxiliary message buffer
DAB\$W_IDI	Word integer	Reserved to Digital
DAB\$W_STATE	Word integer	Value of the current stallpoint
DAB\$L_FLAGS	Longword integer	A bit mask representing options to use when a DATATRIEVE routine is called
DAB\$L_OPTIONS	Longword integer	A bit mask representing the options specified when the DATATRIEVE Interface was initialized
DAB\$W_REC_LEN	Word integer	Length of the record DATATRIEVE is either ready to pass to or to receive from the calling program (for use with the DTR\$GET_PORT and DTR\$PUT_PORT calls)
DAB\$W_VERSION	Word integer	Reserved to Digital
DAB\$W_LEVEL	Word integer	Reserved to Digital
DAB\$B_VER_LETTER	Byte string character	Reserved to Digital
DAB\$W_BASE_LEVEL	Word integer	Reserved to Digital

(continued on next page)

Definitions of the DATATRIEVE Access Block

A.3 Defining the DATATRIEVE Access Block in Other VAX Languages

Table A-4 (Cont.) Fields of the DATATRIEVE Access Block

DAB Field Name	Data Type	Content of Field
DAB\$W_UDK_INDEX	Word integer	The index for a DATATRIEVE or user-defined keyword that the user entered
DAB\$W_COLUMNS_PAGE	Word integer	Reserved to Digital
DAB\$W_TT_CHANNEL	Word integer	The input/output channel for ADT, Guide Mode, and help ¹
DAB\$W_CTLC_CHANNEL	Word integer	The input channel for trapping CTRL/C interrupts
DAB\$L_KEYTABLE_ID	Longword integer	The key table ID for key definitions for the command and prompt keyboards
DAB\$L_COMMAND_KEYBOARD	Longword integer	The virtual keyboard ID that the terminal server uses for command input from the terminal (DATATRIEVE uses the Run-Time Library Screen Management Facility for input from the keyboard)
DAB\$L_PROMPT_KEYBOARD	Longword integer	The virtual keyboard ID that the terminal server uses for input associated with the prompts to the terminal (DATATRIEVE uses separate keyboard IDs for command input and prompting)
DAB\$A_LOG_RAB	Longword integer	The RAB of the .LOG file
DAB\$L_LMF_CONTEXT	Longword integer	Reserved to Digital
DAB\$L_RETURN_VALUE	Longword integer	
DAB\$W_MAIN_WINDOW_X	Word integer	The X value of the current main window
DAB\$W_MAIN_WINDOW_Y	Word integer	The Y value of the current main window
DAB\$REST_OF_DAB	21 bytes (unspecified)	Reserved for future use

¹Form managements (namely FMS, TDMS, DECforms) require a value different from zero to operate.

Definitions of the DATATRIEVE Access Block

A.3 Defining the DATATRIEVE Access Block in Other VAX Languages

A.3.2 Defining the Message Buffers

In addition to the Access Block, DATATRIEVE uses two message buffers to communicate with your program. These message buffers are static character strings, with a recommended length of 80 characters. (You can make the message buffers any length you like. However, if they are shorter than 80 characters, there is the possibility that DATATRIEVE will have to truncate messages to fit them in the message buffer.)

A.3.3 Defining DATATRIEVE Constants

In addition to the access block and message buffers, you may want to use symbolic names for DATATRIEVE stallpoints and options. To use the symbolic names described in this manual, you should define the constants listed in Table A-5.

Table A-5 DATATRIEVE Access Block Constants

DAB Constant	Data Type	Value
Stallpoints		
DTR\$K_STL_CMD	Longword integer	1
DTR\$K_STL_PRMPPT	Longword integer	2
DTR\$K_STL_LINE	Longword integer	3
DTR\$K_STL_MSG	Longword integer	4
DTR\$K_STL_PGET	Longword integer	5
DTR\$K_STL_PPUT	Longword integer	6
DTR\$K_STL_CONT	Longword integer	7
DTR\$K_STL_UDK	Longword integer	8
DTR\$K_STL_END_UDK	Longword integer	9
Initialization Options		
DTR\$K_SEMI_COLON_OPT	Longword integer	1
DTR\$K_UNQUOTED_LIT	Longword integer	16
DTR\$K_SYNTAX_PROMPT	Longword integer	32
DTR\$K_IMMED_RETURN	Longword integer	64
DTR\$K_FORMS_ENABLE	Longword integer	128

(continued on next page)

Definitions of the DATATRIEVE Access Block
A.3 Defining the DATATRIEVE Access Block in Other VAX Languages

Table A-5 (Cont.) DATATRIEVE Access Block Constants

DAB Constant	Data Type	Value
Initialization Options		
DTR\$K_VERIFY	Longword integer	256
DTR\$K_CONTEXT_SEARCH	Longword integer	2048
DTR\$K_HYPHEN_DISABLED	Longword integer	4096
DTR\$K_MORE_COMMANDS	Longword integer	8192
DTR\$K_ABORT	Longword integer	16384
DTR\$K_LOCK_WAIT	Longword integer	32768
Command Options		
DTR\$m_OPT_CMD	Longword integer	1
DTR\$m_OPT_PRMT	Longword integer	2
DTR\$m_OPT_LINE	Longword integer	4
DTR\$m_OPT_MSG	Longword integer	8
DTR\$m_OPT_PGET	Longword integer	16
DTR\$m_OPT_PPUT	Longword integer	32
DTR\$m_OPT_CONT	Longword integer	64
DTR\$m_OPT_UDK	Longword integer	128
DTR\$m_OPT_DTR_UDK	Longword integer	256
DTR\$m_OPT_END_UDK	Longword integer	512
DTR\$m_OPT_UNWIND	Longword integer	1024
DTR\$m_OPT_CONTROL_C	Longword integer	2048
DTR\$m_OPT_STARTUP	Longword integer	4096
DTR\$m_OPT_FOREIGN	Longword integer	8192
DTR\$m_OPT_BANNER	Longword integer	16384
DTR\$m_OPT_REMOVE_CTL	Longword integer	32768
DTR\$m_OPT_KEYDEFS	Longword integer	64536

(continued on next page)

Definitions of the DATATRIEVE Access Block
A.3 Defining the DATATRIEVE Access Block in Other VAX Languages

Table A-5 (Cont.) DATATRIEVE Access Block Constants

DAB Constant	Data Type	Value
User-Defined Keyword Types		
DTR\$K_UDK_SET	Longword integer	1
DTR\$K_UDK_SET_NO	Longword integer	2
DTR\$K_UDK_SHOW	Longword integer	3
DTR\$K_UDK_STATEMENT	Longword integer	4
DTR\$K_UDK_COMMAND	Longword integer	5
String Token Types		
DTR\$K_TOK_TOKEN	Longword integer	1
DTR\$K_TOK_PICTURE	Longword integer	2
DTR\$K_TOK_FILENAME	Longword integer	3
DTR\$K_TOK_COMMAND	Longword integer	4
DTR\$K_TOK_TEST_TOKEN	Longword integer	5
DTR\$K_TOK_LIST_ELEMENT	Longword integer	6
DTR\$K_TOK_TEST_EOL	Longword integer	7

B

DATATRIEVE Message Information

The documentation of DATATRIEVE messages is now stored on-line and may be accessed at the following file specification:

DTR\$LIBRARY:DTRMSG.S.MEM

This file contains information on the severe, error, warning, and informational messages that are used in DATATRIEVE. You can search the file for specific error message information or you can print the file for a hardcopy printout of all DATATRIEVE messages.

For help with error messages when using DATATRIEVE interactively, you can also use the on-line DATATRIEVE help files. If you enter the HELP command and then enter ERROR when prompted for a topic, DATATRIEVE displays a listing of all error messages. If you enter the name of a specific error message, DATATRIEVE displays an explanation of the error and a suggested user action.

When DATATRIEVE displays an error message as a result of some user action, you can type HELP ERROR at the DTR> prompt and DATATRIEVE displays the help text pertaining to that error. HELP ERROR provides information on the last error message you received or on any other message that you specify.

C

Argument Data Types

Each data type implemented for a higher level language uses one of the following VAX data types for procedure parameters and elements of file records.

Data types fall into three categories: atomic, string, and miscellaneous. You can generally pass these data types by immediate value (if 32 bits or less), by reference, or by descriptor. Unless explicitly stated otherwise, all data types represent signed quantities. The unsigned quantities do not allocate space for the sign.

All data types have the prefix `DSC$K_DTYPE_`.

Each of the following tables represents one of the three categories of data types. Table C-1 lists the atomic data types.

Table C-1 Atomic Data Types

Symbol	Code	Name/Description
<code>DSC\$K_DTYPE_Z</code>	0	Unspecified The calling program has specified no data type. The called procedure should assume the argument is of the correct type.
<code>DSC\$K_DTYPE_V</code>	1	Bit An aligned bit string.
<code>DSC\$K_DTYPE_BU</code>	2	Byte logical An 8-bit unsigned quantity.
<code>DSC\$K_DTYPE_WU</code>	3	Word logical A 16-bit unsigned quantity.
<code>DSC\$K_DTYPE_LU</code>	4	Longword logical A 32-bit unsigned quantity.

(continued on next page)

Argument Data Types

Table C–1 (Cont.) Atomic Data Types

Symbol	Code	Name/Description
DSC\$K_DTYPE_QU	5	Quadword logical A 64-bit unsigned quantity.
DSC\$K_DTYPE_OU	25	Octaword logical A 128-bit unsigned quantity.
DSC\$K_DTYPE_B	6	Byte integer An 8-bit signed two's-complement integer.
DSC\$K_DTYPE_W	7	Word integer A 16-bit signed two's-complement integer.
DSC\$K_DTYPE_L	8	Longword integer A 32-bit signed two's-complement integer.
DSC\$K_DTYPE_Q	9	Quadword integer A 64-bit signed two's-complement integer.
DSC\$K_DTYPE_O	26	Octaword integer A 128-bit signed two's-complement integer.
DSC\$K_DTYPE_F	10	F_floating A 32-bit F_floating quantity representing a single-precision number.
DSC\$K_DTYPE_D	11	D_floating A 64-bit D_floating quantity representing a double-precision number.
DSC\$K_DTYPE_G	27	G_floating A 64-bit G_floating quantity representing a double-precision number.
DSC\$K_DTYPE_H	28	H_floating A 128-bit H_floating quantity representing a quadruple-precision number.
DSC\$K_DTYPE_FC	12	F_floating complex An ordered pair of F_floating quantities, representing a single-precision complex number. The lower addressed quantity is the real part; the higher addressed quantity is the imaginary part.
DSC\$K_DTYPE_DC	13	D_floating complex An ordered pair of D_floating quantities, representing a double-precision complex number. The lower addressed quantity is the real part; the higher addressed quantity is the imaginary part.

(continued on next page)

Argument Data Types

Table C–1 (Cont.) Atomic Data Types

Symbol	Code	Name/Description
DSC\$K_DTYPE_GC	29	G_floating complex An ordered pair of G_floating quantities, representing a double-precision complex number. The lower addressed quantity is the real part; the higher addressed quantity is the imaginary part.
DSC\$K_DTYPE_HC	30	H_floating complex An ordered pair of H_floating quantities, representing a quadruple-precision complex number. The lower addressed quantity is the real part; the higher addressed quantity is the imaginary part.
DSC\$K_DTYPE_CIT	31	COBOL intermediate temporary Floating-point data with an 18-digit normalized decimal fraction and a 2-decimal-digit exponent. The fraction is a packed decimal string. The exponent is a 16-bit two's-complement integer.
DSC\$K_DTYPE_VU	34	Bit unaligned The data are 0 to 2**16-1 contiguous bits located arbitrarily with respect to boundaries. See also bit (V) data type. Because additional information is required to specify the bit position of the first bit, this data type can only be used with the unaligned bit string and unaligned bit array descriptors.

The string types listed in Table C–2 are ordinarily described by a string descriptor.

Argument Data Types

Table C–2 String Data Types

Symbol	Code	Name/Description
DSC\$K_DTYPE_T	14	Character-coded text A single 8-bit character (atomic data type) or a sequence of 0 or more 8-bit characters (string data type)
DSC\$K_DTYPE_VT	37	Varying character-coded text
DSC\$K_DTYPE_NU	15	Numeric string, unsigned
DSC\$K_DTYPE_NL	16	Numeric string, left separate sign
DSC\$K_DTYPE_NLO	17	Numeric string, left overpunched sign
DSC\$K_DTYPE_NR	18	Numeric string, right separate sign
DSC\$K_DTYPE_NRO	19	Numeric string, right overpunched sign
DSC\$K_DTYPE_NZ	20	Numeric string, zoned sign
DSC\$K_DTYPE_P	21	Packed decimal string

Table C–3 lists the miscellaneous data types.

Table C–3 Miscellaneous Data Types

Symbol	Code	Name/Description
DSC\$K_DTYPE_ZI	22	Sequence of instructions
DSC\$K_DTYPE_ZEM	23	Procedure entry mask
DSC\$K_DTYPE_DSC	24	Descriptor This data type allows a descriptor to be a data type; thus, levels of descriptors are allowed.

(continued on next page)

Argument Data Types

Table C-3 (Cont.) Miscellaneous Data Types

Symbol	Code	Name/Description
DSC\$K_DTYPE_BPV	32	Bound procedure value A 2-longword entity in which the first longword contains the address of a procedure entry mask and the second longword is the environment value. The environment value is determined in a language-specific manner when the original bound procedure value is generated. When the bound procedure is called, the calling program loads the second longword into R1. When the environment value is not needed, this data type can be passed using the immediate value mechanism. In this case, the argument list entry contains the address of the procedure entry mask and the second longword is omitted.
DSC\$K_DTYPE_BLV	33	Bound label value A two longword entity in which the first longword contains the address of an instruction and the second longword is the language-specific environment value. The environment value is determined in a language specific manner when the original bound label value is generated.
DSC\$K_DTYPE_ADT	35	Absolute date and time

The codes 36 through 191 are reserved to Digital. Codes 192 through 255 are reserved for Digital Computer Special Systems Group and for customers for their own use.

Index

!CMD substitution directive, 3-2
!VAL substitution directive, 3-2

A

Access Block

See DATATRIEVE Access Block

ADT

See Application Design Tool

Application Design Tool, 7-1

channel assignment, 2-13

customizing, 7-1

text file, 7-2

translating, 9-7

Auxiliary message buffer, 2-11, 3-58,
3-82

B

BASIC

DAB, A-1t

INFO inclusion file, 3-49, 3-67

programs

declaring ports, 3-31

DTR\$COMMAND call, 3-5

DTR\$CREATE_UDK call, 3-16

DTR\$DTR call, 3-21

DTR\$GET_STRING call, 3-39

DTR\$INFO call, 3-54

DTR\$INIT call, 3-63

DTR\$LOOKUP call, 3-68

DTR\$PUT_OUTPUT call, 3-77

DTR\$WINDOWS call, 3-96

entering DCL commands from,
3-16

BASIC

programs (cont'd)

entering substitution directives
from, 3-6

initializing DATATRIEVE from,
3-63

simulating interactive

DATATRIEVE, 3-21, 3-96

using message buffers, 2-10

using user-defined keywords,
3-39, 3-53, 3-54

sample programs, A-2

Buffers

See Message buffers

C

C

DAB, A-1t

sample programs, A-2

Call Interface, 1-1

closing, 2-4, 3-25, 3-27

in DECwindows, 3-27

initializing, 2-2, 3-58

messages in, 6-2

Callable DATATRIEVE

compiling programs, 2-4

linking programs, 2-4

Calls to DATATRIEVE, 2-1

and stallpoints, 2-7t

basic steps in, 2-1

DTR\$COMMAND, 3-2

DTR\$CONTINUE, 3-9

DTR\$CREATE_UDK, 3-12

DTR\$DTR, 3-17

Calls to DATATRIEVE (cont'd)

- DTR\$END_UDK, 3-23
- DTR\$FINISH, 3-25
- DTR\$FINISH_WINDOWS, 3-27
- DTR\$GET_PORT, 3-29
- DTR\$GET_STRING, 3-35, 3-52
- DTR\$INFO, 3-41
- DTR\$INIT, 3-58
- DTR\$LOOKUP, 3-65
- DTR\$PORT_EOF, 3-69
- DTR\$PRINT_DAB, 3-73
- DTR\$PUT_OUTPUT, 3-76
- DTR\$PUT_PORT, 3-79
- DTR\$PUT_VALUE, 3-82
- DTR\$UNWIND, 3-85
- DTR\$WINDOWS, 3-92
- how to read format, 3-1

Channel assignment

- DAB\$L_COMMAND_KEYBOARD, 2-14
- DAB\$L_PROMPT_KEYBOARD, 2-14
- DAB\$W_TT_CHANNEL, 2-12
- deassigning channels, 2-13

Closing the Call Interface, 2-4, 3-25, 3-27

- in DECwindows, 3-27

COBOL

- COPY statement and reserved words, 3-30, 3-79
- DAB, A-1t
- INFO inclusion file, 3-49, 3-67
- programs
 - compiling and linking, 2-4
 - declaring condition codes, 2-9
 - displaying print lines, 3-21, 3-96
 - DTR\$COMMAND call, 3-5
 - DTR\$DTR call, 3-21
 - DTR\$GET_PORT call, 3-32
 - DTR\$INIT call, 3-63
 - DTR\$WINDOWS call, 3-96
 - using ports, 3-32
 - using record buffers, 3-32
- sample programs, A-2

Collections

- dropping records from, 3-51

Collections (cont'd)

- getting information about, 3-49t
- invisible, 3-49

Command processing, 2-17

- with DTR\$COMMAND call, 2-3, 2-17, 3-2
- with DTR\$CONTINUE call, 2-17, 3-9
- with DTR\$DTR call, 2-17
- with DTR\$PUT_VALUE call, 2-17
- with DTR\$UNWIND call, 2-17
- with DTR\$WINDOWS call, 2-17

Command recall, 2-14

- and DTR\$COMMAND_LINES, 2-14
- and DTR\$PROMPT_LINES, 2-14

Compiling

- DATATRIEVE functions, 4-5
- message files, 6-10
- programs with DATATRIEVE calls, 2-4

Condition codes

- continuing from, 2-19
- DAB\$L_CONDITION, 2-9, 3-61
- how to interpret, 2-19
- how to use, 2-9
- with DTR\$GET_STRING call, 3-37
- with DTR\$UNWIND call, 3-85

Constants

- definition of, A-7

Context searcher

- enabling, 3-60

Control breaks

- handling with DTR\$DTR call, 3-19

Customizing

- ADT, 7-1
- DATATRIEVE keywords, 8-4
- DATATRIEVE text, 8-1
- dates, 8-3
- files used in, 1-4
- Guide Mode, 7-5
- Help text, 5-1
- interactive DATATRIEVE, 3-22
- messages, 6-1
- syntax prompts, 8-1

D

DAB

See DATATRIEVE Access Block

DAB\$L_COMMAND_KEYBOARD field,
2-14, 3-20

DAB\$L_CONDITION field, 2-9, 3-61
with DTR\$UNWIND call, 3-85

DAB\$L_KEYTABLE_ID field, 3-20

DAB\$L_OPTIONS field, 2-15, 3-59, 3-61
enabling prompting, 2-15

DAB\$L_PROMPT_KEYBOARD field,
2-14, 3-20

DAB\$W_AUX_BUF_LEN field, 2-11

DAB\$W_AUX_LEN field, 2-11

DAB\$W_KEYTABLE_ID field, 2-14

DAB\$W_MSG_BUF_LEN field, 2-11

DAB\$W_MSG_LEN field, 2-11

DAB\$W_REC_LEN field, 2-12

DAB\$W_STATE field, 2-12, 2-19
after DTR\$COMMAND call, 2-4
possible values for, 2-12t

DAB\$W_TT_CHANNEL field, 2-12,
3-20, 3-94

DAB\$W_UDK_INDEX field, 2-15, 3-12
with DATATRIEVE keywords, 2-16

Data Manipulation Facility (DMF), 1-2

Data types

VAX, 4-10t, C-1

DATATRIEVE Access Block, 2-1, 2-2,
2-8, A-3f

AUX_BUFFER, 2-10

contents of, 2-2, 2-8, A-5t

DAB\$L_COMMAND_KEYBOARD,
2-14, 3-20

DAB\$L_CONDITION, 2-9

DAB\$L_KEYTABLE_ID, 2-14, 3-20

DAB\$L_OPTIONS, 2-15, 3-61

DAB\$L_PROMPT_KEYBOARD, 2-14,
3-20

DAB\$W_REC_LEN, 2-12

DAB\$W_STATE, 2-12

DAB\$W_TT_CHANNEL, 2-12, 3-20,
3-94

DATATRIEVE Access Block (cont'd)

DAB\$W_UDK_INDEX, 2-15, 3-12

defining, A-3

definition files, A-1t

fields in, 2-8t

fields used with user-defined keywords,
2-15

initializing, 3-25, 3-27, 3-61

location of files, 2-2, A-1

message buffers, 2-10

MSG_BUFFER, 2-10

printing, 2-20, 3-73

reusing, 3-25, 3-27, 3-61

DATATRIEVE keywords, 2-16

See also User-defined keywords

index values, 2-16t, 3-13

DATATRIEVE objects

getting information about, 3-41, 3-65

DATATRIEVE text

customizing, 8-1

dates, 8-3

editing, 8-5

keywords, 8-4

syntax prompts, 8-1

DATATRIEVE-defined keywords

synonyms for, 2-15

Dates

customizing the format of, 8-3

DCL

See Digital Command Language

Deassigning channels, 2-13, 2-14

Debugging programs, 3-73

DTR\$PRINT_DAB call, 2-20

DECLARE PORT statement, 2-17, 3-30,
3-79

DECwindows calls

DTR\$FINISH_WINDOWS, 3-27

DTR\$WINDOW_MSG, 3-88

DTR\$WINDOW_OUTPUT, 3-90

DECwindows terminal server

See also Terminal server

calling from a program, 3-92

ending a session, 3-27

invoking with DTR\$WINDOWS call,
2-3

DEFINE PORT command, 2-17, 3-30, 3-79
 Digital Command Language (DCL)
 enabling from DATATRIEVE, 3-14, 3-16
 Directives
 See FAO directives
 DMF
 See Data Manipulation Facility (DMF)
 Domains
 getting information about, 3-49t, 3-67
 DTR\$COMMAND call, 2-3, 3-2, 3-60
 and DTR\$K_STL_CMD stallpoint, 2-3, 2-6
 and DTR\$K_STL_UDK stallpoint, 2-6
 DTR\$K_MORE_COMMANDS option, 3-4
 from BASIC, 3-5
 from COBOL, 3-5
 from FORTRAN, 3-5
 from Pascal, 3-6
 to define a port, 3-30
 DTR\$COMMAND_LINES logical name, 2-14
 DTR\$CONTINUE call, 2-17, 3-9
 and DTR\$K_STL_CONT stallpoint, 2-6
 and DTR\$K_STL_LINE stallpoint, 2-6
 and DTR\$K_STL_MSG stallpoint, 2-6
 from FORTRAN, 3-10
 from Pascal, 3-10
 DTR\$CREATE_UDK call, 2-18, 3-12
 from BASIC, 3-16
 from FORTRAN, 3-52
 specifying indexes, 2-15
 DTR\$DTR call, 3-17
 and DTR\$K_STL_CMD stallpoint, 2-3, 2-6
 and DTR\$K_STL_CONT stallpoint, 2-6
 and DTR\$K_STL_LINE stallpoint, 2-6
 and DTR\$K_STL_MSG stallpoint, 2-6
 and DTR\$K_STL_PRMPPT stallpoint, 2-6
 DTR\$DTR call (cont'd)
 DTR\$M_OPT_CONTROL_C option, 3-19
 DTR\$M_OPT_FOREIGN option, 3-16
 DTR\$M_OPT_STARTUP option, 3-20, 3-21
 DTR\$M_OPT_UNWIND option, 3-19
 from BASIC, 3-21
 from COBOL, 3-21
 from FORTRAN, 3-21
 options, 3-17t, 3-18t
 processing user-defined keywords, 2-15, 2-18, 3-13
 DTR\$END_UDK call, 2-18, 3-23
 and DTR\$K_STL_END_UDK stallpoint, 2-7
 and DTR\$K_STL_UDK stallpoint, 2-6, 3-13
 from FORTRAN, 3-24
 DTR\$FINISH call, 2-4, 3-25
 from FORTRAN, 3-26
 DTR\$FINISH_WINDOWS call, 2-4, 3-27
 from FORTRAN, 3-28
 DTR\$GET_PORT call, 3-29, 3-31
 and DTR\$K_STL_PGET stallpoint, 2-6, 2-17
 from COBOL, 3-32
 DTR\$GET_STRING call, 2-18, 3-13, 3-35
 and DTR\$K_STL_UDK stallpoint, 2-6
 DTR\$K_TOK_TOKEN token type, 3-37
 from BASIC, 3-39
 from FORTRAN, 3-37, 3-52
 token types, 3-35
 DTR\$INFO call, 2-18, 3-41
 DTR\$K_INF_COL_INVISIBLE option, 3-49
 from BASIC, 3-54
 from FORTRAN, 3-51
 INFO inclusion file, 3-49
 options, 3-42t
 DTR\$INIT call, 2-2, 3-58
 DTR\$K_SYNTAX_PROMPT, 2-15
 enabling prompting, 2-15

DTR\$INIT call (cont'd)
 from BASIC, 3-63
 from COBOL, 3-63
 from FORTRAN, 3-62
 from Pascal, 3-64
 options, 2-15, 3-59
DTR\$K_ABORT option, 3-60
DTR\$K_CONTEXT_SEARCH option,
 3-60
DTR\$K_FORMS_ENABLE option, 3-59
DTR\$K_HYPHEN_DISABLED option,
 3-60
DTR\$K_IMMED_RETURN option, 3-9,
 3-59
 and DTR\$K_STL_CONT stallpoint,
 2-6
DTR\$K_INF_COL_INVISIBLE option,
 3-49
DTR\$K_LOCK_WAIT option, 3-60
DTR\$K_MORE_COMMANDS option,
 3-4, 3-60
DTR\$K_SEMI_COLON_OPT option,
 3-59
DTR\$K_STL_CMD stallpoint, 2-3, 2-5,
 3-3, 3-13, 3-23, 3-49, 3-60, 3-61,
 3-67
 after initialization, 2-2
DTR\$K_STL_CONT stallpoint, 2-6, 3-9
 and DTR\$K_IMMED_RETURN option,
 2-6
DTR\$K_STL_END_UDK stallpoint, 2-7,
 3-13, 3-23, 3-49, 3-67
DTR\$K_STL_LINE stallpoint, 2-6, 3-9
DTR\$K_STL_MSG stallpoint, 2-6, 3-9,
 3-31, 3-80, 3-85
DTR\$K_STL_PGET stallpoint, 2-6, 2-12,
 2-17, 3-31
DTR\$K_STL_PPUT stallpoint, 2-6, 2-17,
 2-18, 3-69, 3-80
DTR\$K_STL_PRMPPT stallpoint, 2-6,
 2-17, 3-82
DTR\$K_STL_UDK stallpoint, 2-6, 3-3,
 3-13, 3-23, 3-37, 3-49, 3-67
DTR\$K_SYNTAX_PROMPT option, 2-15,
 3-59
DTR\$K_TOK_TEST_TOKEN token type,
 3-37
DTR\$K_TOK_TOKEN token type, 3-37
DTR\$K_UDK_STATEMENT context,
 3-13
DTR\$K_UNQUOTED_LIT option, 3-59
DTR\$K_VERIFY option, 3-59
DTR\$LOOKUP call, 2-18, 3-65
 from BASIC, 3-68
 from FORTRAN, 3-51, 3-68
 INFO inclusion file, 3-67
 object types, 3-65
 with DTR\$INFO call, 3-51
DTR\$M_OPT_CMD option, 2-19
DTR\$M_OPT_CONTROL_C option, 3-19
DTR\$M_OPT_FOREIGN option, 3-16
DTR\$M_OPT_KEYDEFS option, 3-18t
DTR\$M_OPT_STARTUP option, 3-20,
 3-21, 3-94, 3-96
DTR\$M_OPT_UDK option, 2-18, 3-13
DTR\$M_OPT_UNWIND option, 3-19,
 3-94
DTR\$PORT_EOF call, 3-69
 and DTR\$K_STL_PPUT stallpoint,
 2-6, 2-18, 3-80
DTR\$PRINT_DAB call, 2-20, 3-73
 from FORTRAN, 3-74
DTR\$PROMPT_LINES logical name,
 2-14
DTR\$PUT_OUTPUT call, 2-18, 3-76
 from BASIC, 3-77
DTR\$PUT_PORT call, 3-79
 and DTR\$K_STL_PPUT stallpoint,
 2-6, 2-17
 from FORTRAN, 3-69
DTR\$PUT_VALUE call, 2-17, 3-82
 and DTR\$K_STL_PRMPPT stallpoint,
 2-6, 3-82
 contents of message buffers, 3-82
 from FORTRAN, 3-83
DTR\$UNWIND call, 2-17, 3-85
 condition codes, 3-85

DTR\$WINDOWS call, 2-3, 3-92
 and DTR\$K_STL_CMD stallpoint, 2-3, 2-6
 and DTR\$K_STL_CONT stallpoint, 2-6
 and DTR\$K_STL_LINE stallpoint, 2-6
 and DTR\$K_STL_MSG stallpoint, 2-6
 DTR\$M_OPT_FOREIGN option, 3-16
 DTR\$M_OPT_STARTUP option, 3-94, 3-96
 DTR\$M_OPT_UNWIND option, 3-94
 from BASIC, 3-96
 from COBOL, 3-96
 from FORTRAN, 3-95
 options, 3-92t, 3-93t
 processing user-defined keywords, 2-18, 3-13
 DTR\$WINDOW_MSG, 3-88
 DTR\$WINDOW_OUTPUT, 3-90
 DTRADT logical name, 7-5
 DTRHELP logical name, 5-3
 DTRMSGs logical name, 6-11

E

Ending DATATRIEVE sessions, 2-4, 3-25, 3-27
 in DECwindows, 3-27
 Error handling, 2-19, 3-21, 3-95
See also Condition codes
 Error messages
See Messages
 Exiting DATATRIEVE
See Ending DATATRIEVE sessions

F

FAO directives, 6-4, 6-6t
 formatting, 6-8
 in ADT, 7-3
 interpretive, 6-9
 substitution, 6-7
 Fields
 getting information about, 3-49t

Formatted ASCII output
See FAO directives
 Formatting directives, 6-8
 Forms interface
 channel assignment, 2-13
 enabling, 3-59
 getting information about, 3-49t
 FORTRAN
 DAB, A-1t
 INFO inclusion file, 3-49, 3-67
 programs
 defining record buffers, 3-29
 DTR\$COMMAND call, 3-5
 DTR\$CONTINUE call, 3-10
 DTR\$CREATE_UDK call, 3-51
 DTR\$DTR call, 3-21
 DTR\$END_UDK call, 3-24
 DTR\$FINISH call, 3-26
 DTR\$FINISH_WINDOWS call, 3-28
 DTR\$GET_STRING call, 3-37, 3-51
 DTR\$INFO call, 3-51
 DTR\$INIT call, 3-62
 DTR\$LOOKUP call, 3-51, 3-68
 DTR\$PRINT_DAB call, 3-74
 DTR\$PUT_PORT call, 3-69
 DTR\$PUT_VALUE call, 3-83
 DTR\$WINDOWS call, 3-95
 entering commands interactively, 3-5
 forming record streams, 3-8
 handing error messages, 3-21, 3-95
 initializing DATATRIEVE, 3-62
 printing the DAB, 3-74
 processing user-defined keywords, 3-24, 3-26, 3-28, 3-51
 suppressing messages, 3-10
 using ports, 3-69
 sample programs, A-2
 Functions
 adding to DATATRIEVE, 4-1
 creating, 4-4

Functions (cont'd)

- DATATRIEVE function library, 4-5, 4-16
- defining, 4-5
- in BASIC, 4-4
- in FORTRAN, 4-4
- input arguments, 4-7, 4-10t
- linking, 4-16
- Run-Time Library procedures, 4-3
- sample definitions, 4-11
- translating, 9-9
- using, 4-1

G

Guide Mode, 7-1

- channel assignment, 2-13
- commands and statements, 7-6t, 7-7
- customizing, 7-5
- levels, 7-5, 7-7

H

Handling errors, 2-19, 3-21, 3-95

See also Condition codes

Help

- adding text, 5-3
- changing existing text, 5-1
- channel assignment, 2-13
- customizing, 5-1
- Library, 5-1
- modules, 5-2
- replacing, 5-2
- structure of, 5-2
- translating, 9-4

I

Index values

- for DATATRIEVE keywords, 2-16t
- for user-defined keywords, 3-12, 3-13

Information codes for DTR\$INFO call, 3-42t

- definition files, 3-49

Initializing DATATRIEVE, 2-2, 3-58

- from BASIC, 3-63

Initializing DATATRIEVE (cont'd)

- from COBOL, 3-63
- from FORTRAN, 3-62
- from Pascal, 3-64
- options, 2-15, 3-21, 3-59, 3-61, 3-96
- running startup file, 2-3

Interactive DATATRIEVE

See Terminal server

K

Key definitions, 2-14

- DAB\$L_KEYTABLE_ID field, 3-20
- DTR\$M_OPT_KEYDEFS option, 3-18t

Keywords, 2-15

See also DATATRIEVE keywords

See also User-defined keywords

- adding, 3-12
- defining synonyms, 8-4, 9-3
- getting information about, 3-67
- translating, 9-3

L

Linking

- DATATRIEVE shareable image, 4-16
- message files, 6-10
- programs with DATATRIEVE, 2-4

Log files

- creating from a program, 2-18
- writing records to, 3-76

Logical names

- defining in DATATRIEVE, 4-2
- DTR\$COMMAND_LINES, 2-14
- DTR\$PROMPT_LINES, 2-14
- DTRADT, 7-5
- DTRHELP, 5-3
- DTRMSGs, 6-11

M

Message buffers, 2-10, 3-9

- and stallpoints, 2-10
- auxiliary, 2-11, 3-58, 3-82
- contents of, 2-10t, 2-19, 3-61, 3-81

Message buffers (cont'd)

- contents with DTR\$PUT_VALUE call, 3-82
- creating, 2-2
- definition of, A-7
- using in BASIC, 2-10

Message file, 6-3

- compiling, 6-10
- editing, 6-4
- linking, 6-10
- structure of, 6-4

Messages

- codes, 6-1
- continuing after, 2-6
- customizing, 6-1, 6-3
 - examples, 6-5, 6-9
- DTR\$K_STL_MSG stallpoint, 3-9
- FAO directives, 6-6t
- format, 6-1, 6-3t
- in ADT, 7-5
- in interactive DATATRIEVE, 6-2
- in the Call Interface, 6-2
- listing of, B-1
- names, 6-1
- printing from Pascal procedure, 3-10
- severity level, 6-1
- suppressing, 3-10
- translating, 9-6

Modules

- extracting from help libraries, 5-2

O

Objects

- getting information about, 2-18, 3-65

Opening the Call Interface, 2-2, 3-58

Options

- DTR\$K_CONTEXT_SEARCH option, 3-60
- DTR\$K_HYPHEN_DISABLED option, 3-60
- DTR\$K_IMMEDIATE_RETURN option, 3-9, 3-59
- DTR\$K_INF_COL_INVISIBLE option, 3-49

Options (cont'd)

- DTR\$K_MORE_COMMANDS option, 3-4, 3-60
- DTR\$M_OPT_CMD option, 2-19
- DTR\$M_OPT_CONTROL_C option, 3-19
- DTR\$M_OPT_FOREIGN option, 3-16
- DTR\$M_OPT_STARTUP option, 3-20, 3-21, 3-94, 3-96
- DTR\$M_OPT_UDK option, 2-18, 3-13
- DTR\$M_OPT_UNWIND option, 3-19, 3-94
 - for DTR\$DTR call, 3-17t, 3-18t
 - for DTR\$INIT call, 2-15, 3-59
 - for DTR\$WINDOWS call, 3-92t, 3-93t

Options file, 2-4

P

Parsing user-defined keywords, 3-37

- DTR\$GET_STRING call, 2-18, 3-13
- DTR\$K_STL_UDK stallpoint, 2-6

Pascal

- DAB, A-1t
- INFO inclusion file, 3-49, 3-67
- programs
 - DTR\$COMMAND call, 3-6
 - DTR\$CONTINUE call, 3-10
 - DTR\$INIT call, 3-64
 - entering substitution directives, 3-6
 - initializing DATATRIEVE, 3-64
 - printing messages, 3-10
 - sample programs, A-2

Passing commands to DATATRIEVE, 3-2

Passing records, 2-17

- from DATATRIEVE, 2-17, 3-29, 3-30
- to DATATRIEVE, 2-17, 3-79

PL/I

- DAB, A-1t
- sample programs, A-2

Plots

- getting information about, 3-49t

Ports, 3-29

- declaring, 2-17, 3-30
- defining, 2-17, 3-30, 3-79

Ports

- defining (cont'd)
 - with DTR\$COMMAND call, 3-30
- in BASIC, 3-31
- in COBOL, 3-32
- in FORTRAN, 3-69
- passing records from, 2-6, 2-17, 3-79
- size of, 3-31
- storing definitions in the dictionary,
 - 2-17, 3-30
- storing records in, 2-6, 2-17, 3-31
- terminating processing of, 3-69

Print lines, 3-9

- displaying from COBOL, 3-21, 3-96
- displaying from Pascal, 3-10
- DTR\$K_STL_LINE stallpoint, 2-6

Procedures

- Run-Time Library, 4-2

Prompting

- DTR\$K_STL_PRMPPT stallpoint, 3-82
- enabling, 2-15, 3-59

R

- Recalling DATATRIEVE commands, 2-14
 - and DTR\$COMMAND_LINES, 2-14
 - and DTR\$PROMPT_LINES, 2-14

Record buffers

- copying from the dictionary, 3-30
- defining in FORTRAN, 3-29
- size of, 3-31
- using in COBOL, 3-32

Record length

- DAB\$W_REC_LEN, 2-12

Record streams

- forming from FORTRAN program, 3-8

Records

- from DATATRIEVE, 2-17
- locks
 - waiting for, 3-60
- passing to a program, 3-29
- passing to DATATRIEVE, 2-17
- reading, 2-17
- storing, 2-17, 3-31

Remote server, 1-3

Return status

See Condition codes

Run-Time Library procedures, 4-2

Run-Time Library Screen Management Facility (SMG), 2-14

S

Sample programs

- BASIC, A-2
- C, A-2
- COBOL, A-2
- FORTRAN, A-2
- Pascal, A-2
- PL/I, A-2

Servers, 1-2

Severity level of messages, 6-1

SMG

See Run-Time Library Screen Management Facility (SMG)

Stallpoints, 2-1, 2-5

- and calls, 2-7t
- and DAB\$W_STATE, 2-12t
- and message buffer contents, 2-10t
- declaring in a program, 2-12
- DTR\$K_STL_CMD, 2-5, 3-3, 3-13, 3-23, 3-49, 3-60, 3-61, 3-67
- DTR\$K_STL_CONT, 2-6, 3-9, 3-59
- DTR\$K_STL_END_UDK, 2-7, 3-13, 3-23, 3-49, 3-67
- DTR\$K_STL_LINE, 2-6, 3-9
- DTR\$K_STL_MSG, 2-6, 3-3, 3-9, 3-31, 3-80, 3-85
- DTR\$K_STL_PGET, 2-6, 2-12, 3-31
- DTR\$K_STL_PPUT, 2-6, 3-69, 3-80
- DTR\$K_STL_PRMPPT, 2-6, 2-17, 3-82
- DTR\$K_STL_UDK, 2-6, 3-3, 3-13, 3-23, 3-37, 3-49, 3-67
- handled by DTR\$DTR, 3-17t
- handled by DTR\$WINDOWS, 3-92t
- stallpoint numbers, 2-12t

Startup command file, 3-20, 3-94

how to run from calling program, 2-3

Substitution directives

See also FAO directives

entering interactively, 3-6

in command strings, 3-2, 3-3

Synonyms, 8-4, 9-3

for DATATRIEVE keywords, 2-15

for user-defined keywords, 2-16

Syntax prompting

enabling, 2-15

T

Terminal server, 1-2

See also DECwindows terminal server

and channel assignments, 2-13

and DAB\$L_COMMAND_KEYBOARD,
2-14

and DAB\$L_KEYTABLE_ID, 2-14

and DAB\$L_PROMPT_KEYBOARD,
2-14

and DAB\$W_TT_CHANNEL, 2-12

calling from a program, 3-17, 3-92

command recall, 2-14

and DTR\$COMMAND_LINES,
2-14

DECwindows, 3-92

handling control breaks, 3-19

invoking with DTR\$DTR call, 2-3

virtual keyboards, 2-14

Terminal Server

command recall

and DTR\$PROMPT_LINES, 2-14

Terminating processing

with DTR\$FINISH call, 2-4

with DTR\$FINISH_WINDOWS call,
2-4

TERMSERVE.OLB file, 2-5

Text

ADT, 7-1

customizing help, 5-1

dates, 8-3

in DATATRIEVE, 8-1

message, 6-1

syntax prompts, 8-1

translating, 9-8

Token types, 3-35t

See also User-defined keywords, 3-35

Translating DATATRIEVE, 9-1

ADT, 9-7

functions, 9-9

help, 9-4

keywords, 9-3

messages, 9-6

text, 9-8

U

UDKs

See User-defined keywords

User-defined keywords, 3-12, 3-24

See also DATATRIEVE keywords

adding, 3-16

and DAB fields, 2-15

context of, 3-12t, 3-13

DAB\$W_UDK_INDEX, 2-15, 3-12

DTR\$K_STL_UDK stallpoint, 2-6

examples of, 3-14, 3-38

how to use, 2-18, 3-35

in BASIC, 3-39, 3-53

in FORTRAN, 3-24, 3-26, 3-28,
3-51

index values, 3-13

parsing, 3-37

processing with DTR\$DTR, 2-15,

2-18, 3-13

processing with DTR\$WINDOWS,

2-15, 2-18, 3-13

statements, 3-13

synonyms for, 2-16

terminating processing of, 2-7, 3-23

V

VAX data types, 4-10t, C-1

Virtual keyboards, 2-14

key definitions, 2-14

W

Windows

See DECwindows

