VAX DATATRIEVE

Guide to Programming and Customizing

Order No. AA-P863C-TE

digital™

software

Guide to Programming and Customizing

VAX DATATRIEVE Guide to Programming and Customizing

Order No. AA-P863C-TE

November 1987

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**
	MicroVMS
SOFTWARE VERSION:	**VAX DATATRIEVE V4.1**

digital equipment corporation, maynard, massachusetts

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

digital™

Contents

How to Use This Manual

This manual explains how to call the VAX DATATRIEVE software (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, and VAX PL/I. 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 Installation Guide or the Before You Install letter.

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

Structure

This manual contains twelve chapters, three appendixes, and an index:

Chapter 1	Provides an overview of how to customize DATATRIEVE to fit your information management needs.
Chapter 2	Explains how to call DATATRIEVE from within programs written in languages such as FORTRAN, COBOL, and BASIC.

Chapter 3	Contains sample FORTRAN programs.
Chapter 4	Contains sample COBOL programs.
Chapter 5	Contains sample BASIC programs.
Chapter 6	Contains reference material that describes each DATATRIEVE call in alphabetical order.
Chapter 7	Explains how to create functions you can call from within DATATRIEVE.
Chapter 8	Explains how to customize DATATRIEVE help text.
Chapter 9	Explains how to customize DATATRIEVE messages.
Chapter 10	Explains how to customize ADT and Guide Mode.
Chapter 11	Explains how to customize DATATRIEVE text.
Chapter 12	Explains how to translate DATATRIEVE into foreign languages.
Appendix A	Lists the definitions of the DATATRIEVE Access Block in FORTRAN, COBOL, BASIC, PASCAL, and PL/I.
Appendix B	Explains how to access the documentation of DATATRIEVE messages, which is now supplied on line as part of the installation procedure.
Appendix C	Lists the VAX data types.

Related Documents

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

- *VAX DATATRIEVE Handbook*

  Describes how to create DATATRIEVE applications. The manual includes some tutorial information on describing data and creating procedures.

- *VAX DATATRIEVE User's Guide*

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

- *VAX DATATRIEVE Reference Manual*

  Contains reference information for DATATRIEVE.

- *VAX DATATRIEVE Pocket Guide*

  Contains quick-reference information about using DATATRIEVE.

- *VAX DATATRIEVE Guide to Writing Reports*

  Explains how to use the DATATRIEVE Report Writer.

- *VAX DATATRIEVE Guide to Using Graphics*

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

- *VAX DATATRIEVE Release Notes*

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

- *VAX DATATRIEVE Installation Guide*

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

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

Conventions

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

WORD	An uppercase word in a syntax format is a keyword. You must include it in the statement if the clause is used.
word	A lowercase word in a syntax format indicates a syntax element that you supply.
[ ]	Square brackets enclose optional clauses from which you can choose one or none.
{ }	Braces enclose clauses from which you must choose one alternative.

. . .	A horizontal ellipsis means you can repeat the previous item.
. . .	A vertical ellipsis in an example means that information not directly related to the example has been omitted. A vertical ellipsis in a syntax format means you have the option of repeating the element of the syntax on the preceding line.
<CTRL/x>	This symbol tells you to press the CTRL (control) key and hold it down while pressing a letter key.
<RET>	This symbol indicates the RETURN key. Unless otherwise indicated, end all user input lines in examples by pressing the RETURN key.
Color	Color in examples shows user input.
" "	These quotation marks are called double quotation marks.
' '	These quotation marks are called single quotation marks.

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

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

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

References to Products

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

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

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

- VAX ACMS software is referred to as ACMS.
- VAX CDD software is referred to as CDD.
- VAX DATATRIEVE software is referred to as DATATRIEVE.
- VAX DBMS software is referred to as DBMS.
- VAX Rdb/VMS software is referred to as Rdb/VMS.
- VAX TDMS software is referred to as TDMS.
- VIDA software is referred to as VIDA.

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

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

Technical Changes and New Features

This section describes the new features for VAX DATATRIEVE Version 4.1 documented in this manual:

- The DATATRIEVE Access Block (DAB) has several new fields, including:

  DAB$L_KEYTABLE_ID
  DAB$L_COMMAND_KEYBOARD
  DAB$L_PROMPT_KEYBOARD

  See Chapters 2 and 6 for more information on these fields.

- Several improvements were made to the PASCAL DAB definition file for VAX DATATRIEVE Version 4.0. Note that as a result of these improvements, you may have to modify your PASCAL programs that use the DATATRIEVE Call Interface. See Appendix A for more information and a listing of the DAB.PAS definition file.

- The DATATRIEVE installation kit now includes a DAB definition file for VAX PL/I. See Appendix A for a listing of the DAB.PLI definition file.

- When entering DATATRIEVE commands interactively, you can recall previous commands with the CTRL/B and arrow keys. You can specify how many commands DATATRIEVE can recall by defining the logical names DTR$COMMAND_LINES and DTR$PROMPT_LINES. See Chapter 2 for more information on command recall.

- Since Version 4.0, DATATRIEVE no longer inserts continuation characters (hyphens) into the CDD definitions created by callable programs. Prior to Version 4.0, DATATRIEVE inserted spurious continuation characters when a program sent a sequence of definition commands to DATATRIEVE through the Call Interface with multiple calls to DTR$COMMAND. See the description of the DTR$COMMAND call in Chapter 6 for more information on this change.

- DATATRIEVE no longer links against any Run-Time Library (RTL) that DATATRIEVE does not use. For example, DATATRIEVE does not automatically link against the BASIC RTL. When you write a function definition that uses an RTL routine in an object library that DATATRIEVE does not link against, you must edit the DTRBLDxx.COM to include that RTL. See Chapter 7 for more information.
- The DATATRIEVE Call Interface reference material has been moved from Chapter 2 to a new chapter, Chapter 6.

See the Release Notes for a description of all technical changes to DATATRIEVE for Version 4.1. Enter HELP NEW_FEATURES at the DTR> prompt for technical changes for Version 3.0 and all subsequent releases.

Introduction to Programming and Customizing 1

This manual describes how to customize VAX DATATRIEVE to suit your information management needs. It explains how you can:

- 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 online 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, and VAX PL/I.

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

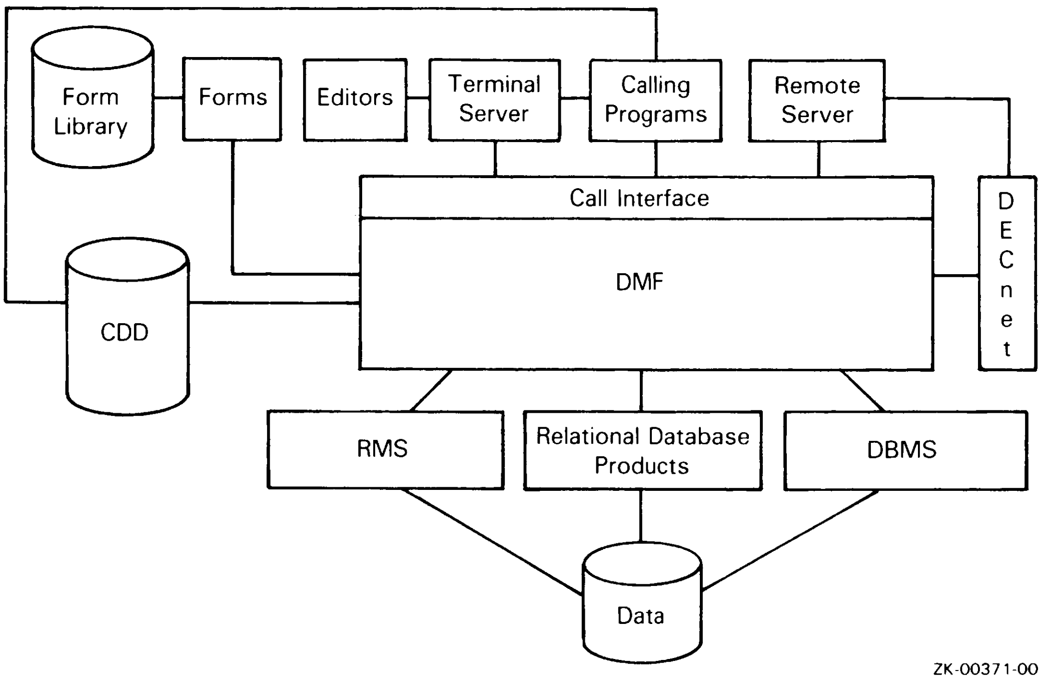

Figure 1-1: VAX DATATRIEVE Architecture

The main part of the DATATRIEVE architecture is the Data Manipulation Facility (DMF). The DMF is the shareable DATATRIEVE image. It parses, optimizes, and executes all commands and statements entered to DATATRIEVE. The DMF uses data definitions stored in the VAX Common Data Dictionary (CDD) 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) RUN command (RUN SYS$SYSTEM:DTR32), 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 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.

1.1.2 How DATATRIEVE Extends Programming Languages

VAX DATATRIEVE enables you to use data from a variety of sources. Records may be stored in a single RMS file, several RMS files (a DATATRIEVE view), or a relational or 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 online 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 SYS$SYSTEM:DTR32
```

You may also want a German version invoked as follows:

```
$ RUN SYS$SYSTEM:DTR32GR
```

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 7 through 12 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.

Writing Programs That Call VAX DATATRIEVE 2

A calling program interacts with DATATRIEVE in much the same way an interactive user does. The program passes command strings, responses to prompts, and structured records to DATATRIEVE. In return, the program receives print lines, messages, and structured records from 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, and PL/I.

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

- DATATRIEVE calls

  Your program communicates with DATATRIEVE by means of calls. A call passes commands, statements, and data from your program to DATATRIEVE.

- DATATRIEVE stallpoints

  After DATATRIEVE executes the commands and statements passed to it in a call, it returns control to your program. DATATRIEVE then waits for further instructions. Those points at which DATATRIEVE waits for further instruction from your program are called stallpoints.

- The DATATRIEVE Access Block

  The DATATRIEVE Access Block (DAB) is a storage section of your program. It contains information that DATATRIEVE and your program use to communicate.

This chapter discusses each of the three concepts separately. Following chapters provide examples of using the DATATRIEVE calls in FORTRAN, COBOL, and BASIC programs and Chapter 6 provides reference material on each DATATRIEVE call in alphabetical order.

The information in this chapter is divided into four parts:

- An overview of DATATRIEVE programming
- A detailed explanation of stallpoints and the DATATRIEVE Access Block
- A discussion of common DATATRIEVE programming tasks
- Sample programs illustrating the basic elements of callable DATATRIEVE in COBOL, BASIC, FORTRAN, and PASCAL.

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 or DTR$DTR.
4. Respond to the subsequent stallpoints and conditions by calling the routines that perform the functions you need.
5. Close the DATATRIEVE Interface by calling DTR$FINISH.

The following sections explain each step in more detail.

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 first part of 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, the DAB contains declarations of integer constants, message buffer sizes, and other information that your program can use.

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

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

Table 2-1 lists the names of the DAB files for each language.

Table 2-1: 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

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.

2.1.3 Issue DATATRIEVE Commands and Statements

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

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 6.)

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.

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 done using DATATRIEVE, it should call DTR$FINISH to finish readied sources and release collections, tables, and variables; for example:

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

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 DATATRIEVE shareable image may be slightly different. Most often, the file name is of the form DTRSHRxx.EXE, with the optional suffix being a two-character version number.

If your program does not call DTR$DTR or DTR$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)

  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$COMMAND or DTR$DTR.

- The prompt stallpoint (DTR$K_STL_PRMPT)

  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$PUT_VALUE or DTR$DTR.

- 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$CONTINUE or DTR$DTR.

- The message stallpoint (DTR$K_STL_MSG)

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

- 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$CONTINUE or DTR$DTR.

  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 6 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$CREATE_UDK. To continue, use the call DTRGET_STRING, DTRCOMMAND, or DTR$END_UDK.

- 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 2-2 summarizes the relationship of stallpoints and calls.

Table 2-2: 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$DTR
DTR$K_STL_PRMPT	It is waiting for the user to enter a value in response to a prompt.	DTR$PUT_VALUE DTR$DTR
DTR$K_STL_LINE	It has a print line for the program to display.	DTR$CONTINUE DTR$DTR
DTR$K_STL_MSG	It has a message for the program to display.	DTR$CONTINUE DTR$DTR
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$DTR
DTR$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

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, the DAB contains 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 2-3 lists the fields of the DAB that you can use and a brief explanation of the field's purpose.

Table 2-3: 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$W_REC_LEN	Length of the record DATATRIEVE passes to the program
DAB$W_TT_CHANNEL	Number of the input/ouput channel for ADT, TDMS forms, Guide Mode, and help

(continued on next page)

Table 2-3: Useful DAB Fields (Cont.)

DAB Field	Contents of Field
DAB$W_COMMAND_KEYBOARD	Keyboard identifier for terminal server command input
DAB$W_PROMPT_KEYBOARD	Keyboard identifier for terminal server input in response to prompts
DAB$W_KEYTABLE_ID	Key table identifier for input key definitions
DAB$W_UDK_INDEX	Number of the current user-defined keyword

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 of this manual for information on how to access the online message documentation.

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 2-4 shows the content of the message buffer corresponding to each stallpoint.

Table 2-4: Content of Message Buffer

Stallpoint	Content of Message Buffer
DTR$K_STL_CMD	Prompt to terminal, for example DTR>
DTR$K_STL_PRMPT	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.

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.

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 2-5 shows the number corresponding to each stallpoint.

Table 2-5: Stallpoint Values

Value of DAB$W_STATE	Stallpoint
1	DTR$K_STL_CMD
2	DTR$K_STL_PRMPT
3	DTR$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 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 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 DATATRIEVE and TDMS Interface.

If your program does not call the DATATRIEVE terminal server and you want to use TDMS 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.

In addition to assigning a channel for ADT, TDMS forms, Guide Mode, and help, the DATATRIEVE terminal server uses the Run-Time Library Screen Management Facility (SMG) to perform command line input. 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 Handbook* 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. 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 6 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$DTR to simulate interactive DATATRIEVE. In the program, you can use the DTR$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.

Some DATATRIEVE keywords are defined internally in the same way as a user-defined keyword. These keywords are called DATATRIEVE-defined keywords. When the user enters a DATATRIEVE-defined keyword, DATATRIEVE passes its number in DAB$W_UDK_INDEX. Table 2-6 shows the index number associated with each DATATRIEVE-defined keyword.

Table 2-6: DATATRIEVE-Defined Keywords

Keyword	Index
EDIT	-4
ADT	-5
GUIDE	-6
EXIT	-7
@	-8
OPEN	-9
CLOSE	-10

(continued on next page)

Table 2-6: DATATRIEVE-Defined Keywords (Cont.)

Keyword	Index
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

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. See Chapters 3 through 5 for extended examples of DATATRIEVE programming. See Chapter 6 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$DTR returns control to DATATRIEVE until DATATRIEVE reaches the stallpoint you specify. By calling DTR$DTR, 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$K_STL_PRMPT) and returns control to your program. DATATRIEVE cannot continue until you return a value with DTR$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 CDD.
- 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

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 DTRGET_PORT, DTRPORT_EOF, and DTR$PUT_PORT calls in Chapter 6 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 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 6 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 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 6 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 6 for more information.

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:

- 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 simply whether the call is valid and whether DATATRIEVE is able to intepret 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 6 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 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. The following BASIC subroutine evaluates both the return status and the stallpoint to identify possible errors:

```
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_BUFF
            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:

```
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
```

2.5 Sample Programs

The following FORTRAN, COBOL, BASIC, PASCAL, and PL/I programs show some of the basic elements of calling DATATRIEVE. Each program performs the same function: letting a user unfamiliar with DATATRIEVE display data from three sample DATATRIEVE domains.

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 Handbook* for information on running the NEWUSER.COM procedure to set up the sample databases.

FORTRAN

```
C Include the DATATRIEVE Access Block.

        INCLUDE 'DTR$LIBRARY:DAB'

C Declare a variable to receive user input.

        CHARACTER*9  DOMAIN

C Initialize the DATATRIEVE Call Interface.

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

C After DATATRIEVE executes this call, it stalls at the command
C stallpoint waiting for a command. This situation is the same
C as when interactive DATATRIEVE displays its DTR> prompt.

C Use a Run-Time Library procedure to clear the screen.

        CALL LIB$ERASE_PAGE (1,1)

C Prompt the user to choose a domain and read in the domain name.

9       WRITE (6, 10)
10      FORMAT (' The following domains are available:'
        1       //' OWNERS'
        2        /' PERSONNEL'
        3        /' YACHTS')

20      WRITE (6,30)
30      FORMAT(/' Enter the name of the domain you want to see: ',$)

        READ (5, 40) DOMAIN
40      FORMAT (A)

C DATATRIEVE is at the command stallpoint.
C Use the call DTR$COMMAND to pass a READY command to DATATRIEVE.
C
C Use the substitution directive !CMD to pass the domain name
C the user selected.

        CALL DTR$COMMAND (DAB, 'READY !CMD', DOMAIN)

C Each time DATATRIEVE attempts to execute a command,
C it puts a condition code in the DAB$L_CONDITION field of the DAB.
C
C If DATATRIEVE successfully executes the READY command, it puts
C the condition DTR$_SUCCESS in DAB$L_CONDITION.
C
C To check for errors, examine the contents of DAB$L_CONDITION.
```

(continued on next page)

```
        IF (DAB$L_CONDITION .NE. %LOC(DTR$_SUCCESS)) THEN

C If the READY command was not completed successfully, DATATRIEVE
C stores an error message in the DAB$A_MSG_BUF field and stalls
C at the message stallpoint.
C
C The call DTR$DTR transfers control to DATATRIEVE. The DTR$M_OPT_CMD
C parameter instructs DATATRIEVE to return control to the
C program when DATATRIEVE is at the command stallpoint.
C

                CALL DTR$DTR (DAB, DTR$M_OPT_CMD)

C DATATRIEVE displays the error message. It then returns control to
C the program and waits for the next command.

C Reprompt the user for a valid domain name.

                WRITE (6, 50)
50              FORMAT (/X, 'Try again: ',$)
                GO TO 9
        END IF

C DATATRIEVE has successfully executed the READY command and has
C a message for the program: "Statement successfully completed."
C
C DATATRIEVE now stalls at the message stallpoint.
C
C To continue passing commands to DATATRIEVE, use the call DTR$CONTINUE.

        CALL DTR$CONTINUE (DAB)

C DATATRIEVE is now at the command stallpoint, ready for the
C next command.
C
C Pass a PRINT domain-name command to DATATRIEVE.

        CALL DTR$COMMAND (DAB, 'PRINT !CMD', DOMAIN)

C DATATRIEVE now has print lines for the program. It is at the
C print line stallpoint.
C
C Use the call DTR$DTR to transfer control to DATATRIEVE. DATATRIEVE
C displays the record stream and returns control to the program
C when it is ready for the next command.

        CALL DTR$DTR (DAB, DTR$M_OPT_CMD)

C End the Call Interface.

        CALL DTR$FINISH (DAB)

        END
```

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

******************************************************************
* Copy in the DAB and set up program variables.                 *
******************************************************************

COPY "DTR$LIBRARY:DAB.LIB".

01 STACK_SIZE PIC 99 COMP VALUE 100.
01 LINE_NO PIC 9 USAGE IS COMP VALUE 1.
01 COL_NO PIC 9  USAGE IS COMP VALUE 1.
01 DOMAIN PIC X(9).
01 COMMAND_LINE PIC X(80).
******************************************************************
PROCEDURE DIVISION.
010-INITIALIZE-THE-INTERFACE.

        CALL "DTR$INIT" USING DAB STACK_SIZE
                        BY DESCRIPTOR MSG_BUFF AUX_BUFF
                        BY REFERENCE DTR$K_SEMI_COLON_OPT.

******************************************************************
* After DATATRIEVE executes this call, it stalls at the          *
* command stallpoint and waits for a command. The situation      *
* is the same as when DATATRIEVE displays its DTR> prompt.       *
******************************************************************

* Use a Run-Time Library procedure to clear the screen.

        CALL "LIB$ERASE_PAGE" USING LINE_NO, COL_NO.

020-DISPLAY-CHOICES.

        DISPLAY "The following domains are available:".
        DISPLAY "".
        DISPLAY " OWNERS".
        DISPLAY " PERSONNEL".
        DISPLAY " YACHTS".

030-READY-THE-DOMAIN.
******************************************************************
* Prompt the user to choose a domain and read in the domain name. *
******************************************************************

        DISPLAY "".
        DISPLAY "Enter the name of the domain you want to see: ".
        ACCEPT DOMAIN.
```

(continued on next page)

```
******************************************************************
* DATATRIEVE is at the command stallpoint. Use the call         *
* DTR$COMMAND to pass a READY command to DATATRIEVE.            *
* Use the substitution directive !CMD to pass the domain        *
* name the user selected.                                       *
******************************************************************

        MOVE "READY !CMD" TO COMMAND_LINE.
        CALL "DTR$COMMAND" USING DAB
                           BY DESCRIPTOR COMMAND_LINE DOMAIN.

******************************************************************
* The call DTR$DTR transfers control to DATATRIEVE. The          *
* DTR$M_OPT_CMD parameter instructs DATATRIEVE to return         *
* control to the program when DATATRIEVE is at the command       *
* stallpoint. If there is an error in readying the domain,       *
* DATATRIEVE prints the error message and returns control.       *
******************************************************************

        CALL "DTR$DTR" USING DAB DTR$M_OPT_CMD.

******************************************************************
* Each time DATATRIEVE attempts to execute a command, it puts    *
* a condition code in the DAB$L_CONDITION field of the DAB.      *
* If DATATRIEVE successfully executes the READY command, it      *
* puts the condition DTR$_SUCCESS in DAB$L_CONDITION.            *
* To check for errors, examine the contents of DAB$L_CONDITION.  *
******************************************************************

        IF DAB$L_CONDITION NOT = DTR$_SUCCESS THEN
                DISPLAY "Try again."
                GO TO 030-READY-THE-DOMAIN.

040-PRINT-THE-RECORD-STREAM.

******************************************************************
* DATATRIEVE is at the command stallpoint. Pass a PRINT          *
* domain-name statement to DATATRIEVE.                           *
******************************************************************

        MOVE "PRINT !CMD" TO COMMAND_LINE.
        CALL "DTR$COMMAND" USING DAB
                           BY DESCRIPTOR COMMAND_LINE DOMAIN.

******************************************************************
* DATATRIEVE now has print lines for the program. It is at the   *
* print line stallpoint. Use the call DTR$DTR to transfer        *
* control to DATATRIEVE. DATATRIEVE displays the record stream   *
* and returns control when it is ready for the next command.     *
******************************************************************

        CALL "DTR$DTR" USING DAB DTR$M_OPT_CMD.

999-END-THE-INTERFACE.
        CALL "DTR$FINISH" USING DAB.
        STOP RUN.
```

BASIC

```
1       ! Include the DATATRIEVE Access Block.

100     %INCLUDE "DTR$LIBRARY:DAB"

        ! Declare variables.

        DECLARE STRING DOMAIN

        ! Initialize the DATATRIEVE Call Interface.

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

        ! Use a Run-Time Library procedure to clear the screen.

        CALL LIB$ERASE_PAGE BY REF (1%,1%)

        ! Prompt the user to choose a domain and read in the domain name.

        PRINT "The following domains are available:"
        PRINT ""
        PRINT "OWNERS"
        PRINT "PERSONNEL"
        PRINT "YACHTS"

300     PRINT ""
        PRINT "Enter the name of the domain you want to see";
        INPUT DOMAIN

        ! Use the call DTR$COMMAND to pass a READY command to DATATRIEVE.
        ! Use the substitution directive CMD to pass the
        ! selected domain name.

        CALL DTR$COMMAND (DAB BY REF, "READY !CMD", DOMAIN)

        ! Each time DATATRIEVE attempts to execute a command,
        ! it puts a condition code in the DAB$L_CONDITION field
        ! of the DAB.

        ! If DATATRIEVE successfully executes the READY command, it puts
        ! the condition DTR$_SUCCESS in DAB$L_CONDITION.

        ! To check for errors, examine the contents of DAB$L_CONDITION.

        IF DAB$L_CONDITION <> DTR$_SUCCESS THEN

        ! If the READY command was not completed successfully, DATATRIEVE
        ! stores an error message in the DAB$A_MSG_BUF field and stalls
        ! at the message stallpoint.

        ! The call DTR$DTR transfers control to DATATRIEVE. The
        ! DTR$M_OPT_CMD argument to this call instructs DATATRIEVE to
        ! return control to the program when DATATRIEVE is ready for the
        ! next command.
```

(continued on next page)

```
! DATATRIEVE displays its error message and returns control to the
! program.

        CALL DTR$DTR BY REF (DAB, DTR$M_OPT_CMD)

! Reprompt the user for a valid domain name.

        PRINT "Try again."
        GO TO 300
END IF

! DATATRIEVE has successfully executed the READY command and has
! a message for the program: "Statement successfully completed."

! DATATRIEVE now stalls at the message stallpoint.

! To recover from this stallpoint and continue passing commands to
! DATATRIEVE, use the call DTR$CONTINUE.

CALL DTR$CONTINUE BY REF (DAB)

! Pass the PRINT domain-name statement to DATATRIEVE.

CALL DTR$COMMAND (DAB BY REF, "PRINT !CMD", DOMAIN)

! DATATRIEVE now has print lines for the program. It is at the
! print line stallpoint.
!
! Use the call DTR$DTR to transfer control to DATATRIEVE. DATATRIEVE
! displays the record stream and returns control to the program when
! it is ready for the next command.

CALL DTR$DTR BY REF (DAB, DTR$M_OPT_CMD)

! End the Call Interface.

CALL DTR$FINISH BY REF (DAB)

END
```

PASCAL

```
PROGRAM PROGRAM_NAME (INPUT, OUTPUT);

{Include the DATATRIEVE Access Block}

%INCLUDE 'DTR$LIBRARY:DAB'

{Declare a procedure to clear the screen.}

PROCEDURE LIB$ERASE_PAGE (LINE_NO : INTEGER; COL_NO : INTEGER); EXTERN;

PROCEDURE MAIN_ROUTINE;

VAR
    LINE_NO : INTEGER;
    COL_NO  : INTEGER;
    DOMAIN  : VARYING[9]  OF CHAR;
    COMMAND : VARYING[20] OF CHAR;

BEGIN
    LINE_NO := 1;
    COL_NO  := 1;
    LIB$ERASE_PAGE (LINE_NO, COL_NO); {Clear the screen.}

{Display the names of available domains.}

    WRITELN('The following domains are available :');
    WRITELN(' ');
    WRITELN('OWNERS');
    WRITELN('PERSONNEL');
    WRITELN('YACHTS');

    WHILE DAB.DAB$L_CONDITION <> DTR$_SUCCESS DO

{When DATATRIEVE successfully executes a command or statement,}
{it puts the condition DTR$_SUCCESS in DAB$L_CONDITION.}
{Use the content of DAB$L_CONDITION to control the current DO loop.}

    BEGIN
        WRITELN;

{Prompt the user for a domain name and read the name into PARAM1.}

        WRITE       ('Enter the name of the domain you want to see: ');
        READLN      (DOMAIN);

{DATATRIEVE is at the command stallpoint.}
{Use the call DTR$COMMAND to pass a READY command to DATATRIEVE.}
{Use the substitution directive !CMD to pass the selected domain name.}

        COMMAND :=  'READY !CMD;';
        DTR$COMMAND (DAB, COMMAND, DOMAIN);

{If the READY command was not completed successfully, DATATRIEVE}
{has an error message and an error code in DAB$L_CONDITION.}
{DATATRIEVE stalls for further instructions.}
```

(continued on next page)

```
{The call DTR$DTR transfers control to DATATRIEVE. The DTR$M_OPT_CMD}
{argument to this call instructs the terminal server to handle all}
{stallpoints, and return control to the program when DATATRIEVE is}
{ready for the next command.}

        DTR$DTR     (DAB, DTR$M_OPT_CMD);

{DATATRIEVE prints its error message and stalls at the command}
{stallpoint.}

    END;

{The READY command has been executed successfully.}
{DATATRIEVE is at the command stallpoint.}
{Pass a PRINT domain-name statement to DATATRIEVE.}

    COMMAND     := 'PRINT !CMD;';
    DTR$COMMAND (DAB, COMMAND, DOMAIN);

{DATATRIEVE now has print lines for the program and stalls at the}
{print line stallpoint.}

{Use the call DTR$DTR to transfer control to DATATRIEVE. DATATRIEVE}
{displays the record stream and returns control to the program when}
{it is ready for the next command.}

    DTR$DTR     (DAB, DTR$M_OPT_CMD);

END;

BEGIN   {Main Program}

{Initialize the Call Interface.}

DTR$INIT (DAB, 100, MSG_BUFF, AUX_BUFF, DTR$K_SEMI_COLON_OPT);

{After DATATRIEVE executes this call, it stalls at the command}
{stallpoint waiting for a command. This situation is the same as}
{when interactive DATATRIEVE displays its DTR> prompt.}

MAIN_ROUTINE;

DTR$FINISH  (DAB); {End the Call Interface.}

END.
```

```
DTR_EXAMPLE: PROCEDURE OPTIONS(MAIN);

  /*
   * Included files and modules.
   */
  %INCLUDE 'DTR$LIBRARY:DAB.PLI';
  DCL LIB$ERASE_PAGE ENTRY (
      FIXED BINARY(15),
      FIXED BINARY(15))
      RETURNS (FIXED BINARY(31)) OPTIONS(VARIABLE);

  /*
   * Local declarations
   */
  DECLARE DOMAIN CHARACTER(32) VARYING;  /* Domain name */
  DECLARE STATUS FIXED BIN(31);    /* For return status values */
  DECLARE 1 DAB LIKE DTR_ACCESS_BLOCK; /* Local copy of the DAB */
  DECLARE MSG_BUFF CHAR(80);
  DECLARE AUX_BUFF CHAR(20);

  /*
   * Initialize the DATATRIEVE Call Interface.
   */
  STATUS = DTR$INIT(DAB, 100, MSG_BUFF, AUX_BUFF,
                              DTR$M_SEMI_COLON_OPT);

  /*
   * After DATATRIEVE executes this call, it stalls at the command
   * stallpoint waiting for a command. This situation is the same
   * as when interactive DATATRIEVE displays its DTR> prompt.
   *
   * Use a Run-Time Library procedure to clear the screen.
   */
  STATUS = LIB$ERASE_PAGE(1,1);

  /*
   * Prompt the user to choose a domain and read in the domain name.
   */
  PUT EDIT('The following domains are available:')(A);
  PUT EDIT('OWNERS','PERSONNEL','YACHTS')(SKIP,3(SKIP,X,A));
  PUT SKIP(2);

  GET LIST(DOMAIN)
      OPTIONS(PROMPT('Enter the name of the domain you want to see: '));

  /*
   * DATATRIEVE is at the command stallpoint.
   * Use the call DTR$COMMAND to pass a READY command to DATATRIEVE.
   *
   * Use the substitution directive !CMD to pass the domain name
   * the user selected.
   */
  STATUS = DTR$COMMAND(DAB, 'READY !CMD', DOMAIN);
```

(continued on next page)

```
/*
 * Each time DATATRIEVE attempts to execute a command,
 * it puts a condition code in the DAB$L_CONDITION field of the DAB.
 * The status value is also the return value from the function.
 *
 * If DATATRIEVE successfully executes the READY command, it puts
 * the condition DTR$_SUCCESS in DAB$L_CONDITION.
 *
 * To check for errors, examine the contents of DAB$L_CONDITION.
 */
DO WHILE(DAB.DAB$L_CONDITION ^= DTR$_SUCCESS);

     /*
      * If the READY command was not completed successfully, DATATRIEVE
      * stores an error message in the DAB$A_MSG_BUF field and stalls
      * at the message stallpoint.
      *
      * The call DTR$DTR transfers control to DATATRIEVE. The DTR$M_OPT_CMD
      * parameter instructs DATATRIEVE to return control to the
      * program when DATATRIEVE is at the command stallpoint.
      */
     STATUS = DTR$DTR(DAB, DTR$M_OPT_CMD);

     /*
      * DATATRIEVE displays the error message. It then returns control to
      * the program and waits for the next command.
      *
      * Reprompt the user for a valid domain-name.
      */
     GET LIST(DOMAIN) OPTIONS(PROMPT('Try again: '));

     /*
      * Retry the operation.
      */
     STATUS = DTR$COMMAND(DAB, 'READY !CMD', DOMAIN);

     END;

/*
 * DATATRIEVE has successfully executed the READY command and has
 * a message for the program: "Statement successfully completed."
 *
 * DATATRIEVE now stalls at the message stallpoint.
 *
 * To continue passing commands to DATATRIEVE, use the call DTR$CONTINUE.
 */
STATUS = DTR$CONTINUE(DAB);

/*
 * DATATRIEVE is now at the command stallpoint, ready for the
 * next command.
 *
 * Pass a PRINT domain-name command to DATATRIEVE.
 */
STATUS = DTR$COMMAND(DAB, 'PRINT !CMD', DOMAIN);
```

```
/*
 * DATATRIEVE now has print lines for the program. It is at the
 * print line stallpoint.
 *
 * Use the call DTR$DTR to transfer control to DATATRIEVE. DATATRIEVE
 * displays the record stream and returns control to the program
 * when it is ready for the next command.
 */
STATUS = DTR$DTR(DAB, DTR$M_OPT_CMD);

/*
 * End the Call Interface.
 */
STATUS = DTR$FINISH(DAB);

END;
```

Sample FORTRAN Programs 3

This chapter contains sample FORTRAN programs that call DATATRIEVE. These programs show how you can call DATATRIEVE to perform calculations on data, store and retrieve data, and create data management applications for end users. Copies of the programs are in the DTR$LIBRARY directory.

To run these programs you must compile them and then link them with the DATATRIEVE shareable image:

```
$ FORTRAN CORRELATE, PARSE, CALCULATE
$ LINK CORRELATE, PARSE, CALCULATE, DTR/OPT
$ RUN CORRELATE
```

See Chapter 2 for more information on using an options file to link programs that call DATATRIEVE.

3.1 Creating an End-User Interface to DATATRIEVE

The program MENU shows you how to give users access to the DATATRIEVE data management capabilities. MENU enables users unfamiliar with DATATRIEVE to display, store, modify, and report data managed by DATATRIEVE.

MENU calls the subroutine MESSAGE to suppress the display of some DATATRIEVE messages and to display all other messages and print lines. The subroutine MESSAGE is listed as an example of the DTR$CONTINUE command in Chapter 6.

In addition, MENU calls the following subroutines:

ESTABLISH
DISPLAY
SORT
MODIFY
REPORT
STORE
CHOOSE

3.1.1 The Main Program: MENU

```
C            M E N U . F O R
C
C Include the DATATRIEVE Access Block.
C
        INCLUDE 'DTR$LIBRARY:DAB'
C
C Declare variables.
C
        CHARACTER*31 DOMAIN
        INTEGER*2    CHOICE
        INTEGER*2    COL/1/
        INTEGER*2    LINE/1/
        INTEGER      STATUS

C Select initial options.

            INIT_OPTS =   DTR$K_SEMI_COLON_OPT
        1                + DTR$K_UNQUOTED_LIT
        2                + DTR$K_FORMS_ENABLE
C
C Initialize the interface with DATATRIEVE.
C
        STATUS = DTR$INIT (DAB, 100, MSG_BUFF, AUX_BUFF, INIT_OPTS)

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

C Clear the screen.

10       CALL LIB$ERASE_PAGE (LINE, COL)

C Declare a port for the count of records in the collection
C the user establishes.

         CALL DTR$COMMAND (DAB, 'DECLARE PORT PT1 USING ')
         CALL DTR$COMMAND (DAB, '01 NUM PIC 9(4) COMP.;')
         CALL DTR$DTR (DAB, DTR$M_OPT_CMD)

C Prompt the user to choose a domain.

         CALL CHOOSE (DOMAIN)
```

(continued on next page)

```
C Display the menu and prompt for a selection.

50      DO WHILE (.TRUE.)

        CHOICE = 0

          DO WHILE ((CHOICE .LT. 1) .OR. (CHOICE .GT. 8))
              ISTAT = LIB$ERASE_PAGE (LINE, COL)
              WRITE (6, 60)
60      FORMAT (//23X,'MENU'/23X'----'/
        1/10X,'1. Establish a collection of records.'
        2/10X,'2. Display the current collection.'
        3/10X,'3. Sort the current collection.'
        4/10X,'4. Update the current collection.'
        5/10X,'5. Report the current collection.'
        6/10X,'6. Store records in the current domain.'
        7/10X,'7. Choose another domain.'
        8/10X,'8. End this session.'/////
        1/10X'    Enter the number of the operation '
        2/10X'    you wish to perform: ',$)

C Read in the user's choice.

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

        END DO

C Call the appropriate subroutine to handle the choice entered.

        IF (CHOICE .EQ. 1) CALL ESTABLISH (DOMAIN)

        IF (CHOICE .EQ. 2) CALL DISPLAY

        IF (CHOICE .EQ. 3) CALL SORT (DOMAIN)

        IF (CHOICE .EQ. 4) CALL MODIFY (DOMAIN)

        IF (CHOICE .EQ. 5) CALL REPORT (DOMAIN)

        IF (CHOICE .EQ. 6) CALL STORE (DOMAIN)

        IF (CHOICE .EQ. 7) THEN
            CALL DTR$COMMAND (DAB, 'FINISH !CMD', DOMAIN)
            CALL DTR$DTR (DAB, DTR$M_OPT_CMD)
            CALL LIB$ERASE_PAGE (LINE, COL)
            CALL CHOOSE (DOMAIN)
        END IF

C Finish the session with DATATRIEVE and stop the program.

        IF (CHOICE .EQ. 8) THEN
                CALL DTR$FINISH (DAB)
                GO TO 999
        END IF

        END DO

999     END
```

3.1.2 The Subroutine ESTABLISH

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

        SUBROUTINE ESTABLISH (DOMAIN)
        INCLUDE 'DTR$LIBRARY:DAB'
        CHARACTER*31 ATTRIBUTE
        CHARACTER*31 ANSWER
        CHARACTER*31 BOOLEAN
        CHARACTER*31 DOMAIN
        CHARACTER*31 VALUE
        INTEGER*4    NUM_RECS
        INTEGER*2    LINE/1/, COL/1/

C Clear the screen.

        CALL LIB$ERASE_PAGE (LINE, COL)

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

        WRITE (6, 101)
101     FORMAT (' Do you wish to use all the records
        1 in the domain? ',$)
        READ (5, 102)ANSWER
102     FORMAT (A)
        IF (ANSWER(1:1) .EQ. 'Y' .OR. ANSWER(1:1) .EQ. 'y')  THEN
                CALL DTR$COMMAND (DAB,'FIND !CMD;',DOMAIN)
                CALL MESSAGE
                RETURN
        END IF

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

105     CALL DTR$COMMAND (DAB,'SHOW FIELDS !CMD;', DOMAIN)

C Call the terminal server to handle the printing of the fields.
C Return to the program on the DTR$K_STL_CMD stallpoint.

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

(continued on next page)

```
C Prompt the user for a field name, a relational operator, and a value.

        WRITE (6, 110)
110     FORMAT (' The collection will be formed on the basis of'
        1 / ' identifying characteristics you choose. You should'
        2 / ' enter a FIELD, an EQUATION, and an ATTRIBUTE. For'
        3 / ' example, if your domain is EMPLOYEES, you can form'
        4 / ' a collection of:'
        5 / ' EMPLOYEES whose SALARY (FIELD) is'
        6 / ' GT (RELATION) $25,000 (VALUE).'
        7 ///' Enter the FIELD (SALARY, PRICE, DEPARTMENT): ',$)
        READ (5,102) ATTRIBUTE

        WRITE (6, 115)
115     FORMAT (' Enter the RELATION ( EQ, GT, GE, LT, CONTAINING): ',$)
        READ (5, 102) BOOLEAN

        WRITE (6, 120)
120     FORMAT (' Enter the VALUE (non-numeric values should be in
        1 quotes): ' ,$)
        READ (5, 102) VALUE

C Instruct DATATRIEVE to find the desired records.

        CALL DTR$COMMAND
        1  (DAB,'FIND CURRENT WITH !CMD !CMD !CMD;',
        2   ATTRIBUTE, BOOLEAN, VALUE)

C Call the subroutine MESSAGE to handle messages.

        CALL MESSAGE

C Verify that the FIND was completed successfully.

        IF (DAB$L_CONDITION .NE. %LOC(DTR$_SUCCESS)) THEN
                GO TO 105
        END IF

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

        CALL DTR$COMMAND (DAB, 'STORE PT1 USING NUM = COUNT;')

C Display messages.

        IF (DAB$W_STATE .EQ. DTR$K_STL_MSG) THEN
                CALL MESSAGE
        END IF

C Use DTR$GET_PORT call to retrieve the number of records found.

        IF (DAB$W_STATE .EQ. DTR$K_STL_PGET) THEN
                CALL DTR$GET_PORT (DAB, NUM_RECS)
                CALL DTR$DTR (DAB, DTR$M_OPT_CMD)
        END IF
```

```
C If no records were found, notify the user and find all the
C records in the domain.
C Prompt the user to continue or return to the main menu.

        IF (NUM_RECS .EQ. 0) THEN
                CALL DTR$COMMAND (DAB, 'FIND !CMD;',DOMAIN)
                CALL MESSAGE
                WRITE (6, 135)
135             FORMAT (' There are no records that fit.'/' Do you wish
        1 to use all the records in the domain? ',$)
                READ (5,102) ANSWER
                IF (ANSWER(1:1) .EQ. 'Y' .OR. ANSWER(1:1) .EQ. 'y') THEN
                        RETURN
                END IF
                GO TO 105
        END IF

        CALL DTR$DTR (DAB, DTR$M_OPT_CMD)

C Ask if the user wishes to make a subcollection.

160     WRITE (6, 165)
165     FORMAT (' Would you like to establish a sub-collection from the
        1 current collection?')
        READ (5,102) ANSWER
        IF (ANSWER(1:1) .EQ. 'Y' .OR. ANSWER(1:1) .EQ. 'y') THEN
                GO TO 105
        END IF

        RETURN
        END
```

3.1.3 The Subroutine DISPLAY

```
C****************************************************
C                 SUBROUTINE DISPLAY                *
C Displays the current collection of records.       *
C****************************************************

        SUBROUTINE DISPLAY
        INCLUDE 'DTR$LIBRARY:DAB'
        CHARACTER*27 TEXT/'PRESS RETURN TO CONTINUE )'/
        CHARACTER*1  CR
        INTEGER*4    DTR_OPTIONS
        INTEGER*2    E_LINE/1/, E_COL/1/
        INTEGER*2    P_LINE/24/, P_COL/15/

        CALL LIB$ERASE_PAGE (E_LINE,E_COL)

C Select terminal server call options.

            DTR_OPTIONS =
        1            DTR$M_OPT_CMD        ! Return on DTR$K_STL_CMD
        2          + DTR$M_OPT_CONTROL_C  ! Enable Control C handling


C Instruct DATATRIEVE to print the current collection.

        CALL DTR$COMMAND (DAB, 'PRINT ALL')

        CALL DTR$DTR (DAB, DTR_OPTIONS)

C Call a library routine to put a message at the bottom of the screen.

        WRITE (6, *)' '
        WRITE (6, *)' '
        CALL LIB$PUT_SCREEN (TEXT, P_LINE, P_COL)

C Wait until the user has pressed RETURN.

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

        END
```

3.1.4 The Subroutine SORT

```
C*************************************************
C             SUBROUTINE SORT                    *
C Sorts the current collection.                  *
C*************************************************

        SUBROUTINE SORT (DOMAIN)
        INCLUDE 'DTR$LIBRARY:DAB'

        CHARACTER*31 FIELDS
        CHARACTER*31 SORTORDER
        CHARACTER*27 TEXT/'PRESS RETURN TO CONTINUE )'/
        CHARACTER*1  CR
        INTEGER*2    E_LINE/1/, E_COL/1/
        INTEGER*2    P_LINE/24/, P_COL/15/
        LOGICAL      UNSORTED/.TRUE./, NO_ORDER/.TRUE./

C Continue in a loop until a successful SORT statement has been completed.

        DO WHILE (UNSORTED)
            CALL LIB$ERASE_PAGE (E_LINE, E_COL)

C Display the available fields.

            CALL DTR$COMMAND (DAB, 'SHOW FIELDS !CMD', DOMAIN)
            CALL DTR$DTR (DAB, DTR$M_OPT_CMD)
            WRITE (6, 310)
310         FORMAT (' Enter the FIELD by which you wish to sort: ',$)
            READ (5, 1) FIELDS
1           FORMAT (A)

C Establish a sort order.

            DO WHILE (NO_ORDER)
                WRITE (6, 320)
320             FORMAT (' Sort in ascending or descending order
        1(A or D)? ', $)

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

                READ (5, 1) SORTORDER
                IF ((SORTORDER(1:1) .EQ. 'A') .OR.
        1           (SORTORDER(1:1) .EQ. 'a')) THEN

C Sort by the ascending field given by the user.

                    NO_ORDER = .FALSE.
                    CALL DTR$COMMAND (DAB, 'SORT CURRENT BY ASCENDING
        1       !CMD;', FIELDS)
                ELSE
                IF ((SORTORDER(1:1) .EQ. 'D') .OR.
        1        (SORTORDER(1:1) .EQ. 'd')) THEN
```

(continued on next page)

```
C Sort by the descending field given by the user.

                    NO_ORDER = .FALSE.
                    CALL DTR$COMMAND (DAB,
        1               'SORT CURRENT BY DESC !CMD;', FIELDS)
                ELSE
                    WRITE (6, 325)
325                 FORMAT (' Re-enter sorting order')
                END IF

                END IF
            END DO

        NO_ORDER = .TRUE.

C Display error messages.

        CALL DTR$DTR (DAB, DTR$M_OPT_CMD)

        IF (DAB$L_CONDITION .NE. %LOC(DTR$_SUCCESS)) THEN
               CALL LIB$PUT_SCREEN (TEXT, P_LINE, P_COL)
               READ (5, 1) CR
        ELSE
               UNSORTED = .FALSE.
        END IF

       END DO

       UNSORTED = .TRUE.

C  Inform user that sort is complete.

370    WRITE (6, 375)
375    FORMAT (/////26X, 'Sort successfully completed.')
       CALL LIB$PUT_SCREEN (TEXT, P_LINE, P_COL)
       READ (5, 1) CR

       RETURN
       END
```

3.1.5 The Subroutine MODIFY

```
C************************************************
C            SUBROUTINE MODIFY                  *
C Sets up a second menu and modifies records.   *
C************************************************

        SUBROUTINE MODIFY (DOMAIN)
        INCLUDE 'DTR$LIBRARY:DAB'
        CHARACTER*80 ANSWER, CHLINE
        CHARACTER*31 DOMAIN, FIELDS, VALUE
        CHARACTER*27 TEXT/'PRESS RETURN TO CONTINUE )'/
        CHARACTER*1  CR
        INTEGER*4    NUMBER
        INTEGER*2    ERASE_LINE/1/, ERASE_COL/1/
        INTEGER*2    P_LINE/24/, P_COL/15/
        INTEGER*2    IANSWER, NUM
        IANSWER = 1

        DO WHILE (IANSWER .NE. 3)

C Erase the screen.

400     CALL LIB$ERASE_PAGE (ERASE_LINE, ERASE_COL)

C Display the modify submenu.

        WRITE (6, 406)
406     FORMAT (////'    1. One or more fields for one record.'/'
        1    2. One field for all records in the current collection.'/'
        2    3. Return to main menu'///' Enter your choice: ',$)
        READ (5, 407) IANSWER
407     FORMAT (I1)

C Command DATATRIEVE to start with the first record in the
C current collection.

        CALL DTR$COMMAND (DAB, 'SELECT 1;')
        CALL DTR$DTR (DAB, DTR$M_OPT_CMD)
        GO TO (410, 430, 499),IANSWER
        WRITE (6, 408)
408     FORMAT (' Invalid operation...try again.')
        CALL LIB$PUT_SCREEN (TEXT, P_LINE, P_COL)
        READ (5, 401) CR
401     FORMAT(A)
        GO TO 400

C Select records one at a time. Prompt for which
C records the user wishes to modify.

409     CALL DTR$COMMAND (DAB, 'SELECT NEXT;')
```

(continued on next page)

```
C Print any messages.

        IF (DAB$W_STATE  .EQ.  DTR$K_STL_MSG) THEN
                 CALL DTR$DTR (DAB, DTR$M_OPT_CMD)
        END IF

410     CALL LIB$ERASE_PAGE (ERASE_LINE, ERASE_COL)

C Display the selected record.

411     CALL DTR$COMMAND (DAB, 'PRINT;')
        CALL DTR$DTR (DAB, DTR$M_OPT_CMD)

C Inquire if this record needs modification.

        WRITE (6, 412)
412     FORMAT (//' Is this the record you wish to update?'//'
        1 Enter YES, NO, or EXIT : ', $)
        READ (5, 401)ANSWER
        IF (ANSWER(1:1) .EQ. 'E' .OR. ANSWER(1:1) .EQ. 'e') GO TO 400
        IF (ANSWER(1:1) .NE. 'Y' .AND. ANSWER(1:1) .NE. 'y') THEN
                NUMBER = NUMBER +1
                GO TO 409
        END IF
        CALL LIB$ERASE_PAGE (ERASE_LINE, ERASE_COL)

C Show the fields available to the user.

413     CALL DTR$COMMAND (DAB, 'SHOW FIELDS !CMD' ,DOMAIN)
        CALL DTR$DTR (DAB, DTR$M_OPT_CMD)
        WRITE (6, 415)
415     FORMAT (' Which field do you wish to modify: ', $)
        READ (5, 401)FIELDS

C Modify and check for errors.

        CALL DTR$COMMAND (DAB, 'MODIFY !CMD;', FIELDS)

        IF (DAB$W_STATE  .EQ.  DTR$K_STL_MSG) THEN
                CALL DTR$DTR (DAB, DTR$M_OPT_CMD)
                WRITE (6, 418)
418     FORMAT (' DO YOU WANT TO TRY AGAIN? ',$)
                READ (5, 401)ANSWER
                IF ((ANSWER .EQ. 'N') .OR. (ANSWER .EQ. 'n')) THEN
                    GO TO 400
                ELSE
                    GO TO 411
                END IF
        END IF

C Print any messages.

        CALL DTR$DTR (DAB, DTR$M_OPT_CMD)
        CALL LIB$ERASE_PAGE (ERASE_LINE, ERASE_COL)
```

```
C Print the modified record on the screen.

        CALL DTR$COMMAND (DAB, 'PRINT;')
        CALL DTR$DTR (DAB, DTR$M_OPT_CMD)
        WRITE (6, 423)
423     FORMAT (/' Do you wish to modify any more fields
        1 in this record? ', $)

        READ (5, 401) ANSWER
        IF (ANSWER(1:1) .EQ. 'Y' .OR. ANSWER(1:1) .EQ. 'y') GO TO 413
        WRITE (6, 425)
425     FORMAT (/' Do you wish to continue updating records?')
        READ (5, 401) ANSWER
        IF (ANSWER(1:1) .EQ. 'Y' .OR. ANSWER(1:1) .EQ. 'y') THEN
                NUMBER = NUMBER +1
                GO TO 409
        END IF
        GO TO 400

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

430     CALL LIB$ERASE_PAGE (ERASE_LINE, ERASE_COL)
432     WRITE (6, 435)
435     FORMAT (/////' Do you want to:'//'    1. Update all records to
        1 one value.'/'    2. Update all records with an equation '/'
        2          (for example, price = price + 400)'/'
        33. Return to the previous menu. '/
        4/' Enter 1, 2 or 3: '$)
        READ (5, 440)NUM
440     FORMAT (I)
        GO TO (460, 445, 400), NUM

C Prompt for an equation.

445     CALL LIB$ERASE_PAGE (ERASE_LINE, ERASE_COL)

C Show the fields available.

        CALL DTR$COMMAND (DAB, 'SHOW FIELDS')
        CALL DTR$DTR (DAB, DTR$M_OPT_CMD)
        WRITE (6, 450)
450     FORMAT (' Enter the equation you wish to use')
        READ (5, 401) CHLINE

C Update all records in the current collection.

        CALL DTR$COMMAND (DAB, 'MODIFY ALL USING !CMD;', CHLINE)

C Check for errors.

        IF (DAB$W_STATE  .EQ.  DTR$K_STL_MSG) THEN
                CALL DTR$DTR (DAB, DTR$M_OPT_CMD)
        END IF
```

(continued on next page)

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

        IF (DAB$L_CONDITION .NE. %LOC(DTR$_SUCCESS)) THEN
                GO TO 430
        END IF
        CALL LIB$ERASE_PAGE (ERASE_LINE, ERASE_COL)
        WRITE (6, 455)
455     FORMAT (' All records updated.')

C Print the updated collection.

        CALL DTR$COMMAND (DAB, 'PRINT CURRENT;')
        CALL DTR$DTR (DAB, DTR$M_OPT_CMD)
        CALL LIB$PUT_SCREEN (TEXT, P_LINE, P_COL)
        READ (5, 401) CR
        GO TO 400

C Modify one field for all records.

460     CALL LIB$ERASE_PAGE (ERASE_LINE, ERASE_COL)
        CALL DTR$COMMAND (DAB, 'SHOW FIELDS')
        CALL DTR$DTR (DAB, DTR$M_OPT_CMD)
        WRITE (6, 465)
465     FORMAT ('Which field do you wish to update? ', $)
        READ (5, 401) FIELDS

C Issue the MODIFY statement to DATATRIEVE.

        CALL DTR$COMMAND (DAB, 'MODIFY ALL !CMD OF CURRENT;', FIELDS)
        IF (DAB$W_STATE  .EQ.  DTR$K_STL_MSG) THEN
                CALL DTR$DTR (DAB, DTR$M_OPT_CMD)
        END IF

C Check to see if MODIFY was successful.

        IF (DAB$L_CONDITION .NE. %LOC(DTR$_SUCCESS)) THEN
                CALL DTR$COMMAND (DAB, 'SHOW FIELDS')
                CALL DTR$DTR (DAB, DTR$M_OPT_CMD)
                GO TO 430
        END IF

C Prompt for a value.

475     WRITE (6, 477) MSG_BUFF
477     FORMAT (1X, A(DAB$W_MSG_LEN), $)
        READ (5, 401) VALUE
```

```
C Pass the value to DATATRIEVE.

        CALL DTR$PUT_VALUE (DAB, VALUE)
        IF (DAB$W_STATE  .EQ.  DTR$K_STL_MSG) THEN
                CALL DTR$DTR (DAB, DTR$M_OPT_CMD)
        END IF
        IF (DAB$L_CONDITION .NE. %LOC(DTR$_SUCCESS)) THEN
                WRITE (6, 478)
478             FORMAT (' Invalid, try again....'//)
                GO TO 430
        END IF
        CALL LIB$ERASE_PAGE (ERASE_LINE, ERASE_COL)

C Print the modified collection.

        CALL DTR$COMMAND (DAB, 'PRINT CURRENT;')
        CALL DTR$DTR (DAB, DTR$M_OPT_CMD)
        CALL LIB$PUT_SCREEN (TEXT, P_LINE, P_COL)
        READ (5,401) CR

        END DO

499     RETURN

        END
```

3.1.6 The Subroutine REPORT

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

        SUBROUTINE REPORT (DOMAIN)
        INCLUDE 'DTR$LIBRARY:DAB'

        EXTERNAL DTR$_SHOWTEXT
        EXTERNAL DTR$_ERROR
        CHARACTER*80 RPTHDR, CHLINE, ANSWER
        CHARACTER*75 SHOFIELDS(100)
        CHARACTER*31 DOMAIN
        INTEGER*2    LINE/1/, COL/1/

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

505     CALL DTR$COMMAND (DAB,'SHOW FIELDS !CMD;', DOMAIN)
        I=0
        DO WHILE ((DAB$W_STATE  .EQ.  DTR$K_STL_MSG) .AND.
        1      (DAB$L_CONDITION .EQ. %LOC(DTR$_SHOWTEXT)))
                I=I+1
                SHOFIELDS(I) = MSG_BUFF(1:DAB$W_MSG_LEN)
                CALL DTR$CONTINUE (DAB)
        END DO

        CALL DTR$DTR (DAB, DTR$M_OPT_CMD)

C Ask the user if he/she wants to use the whole file or a collection.

        CALL LIB$ERASE_PAGE (LINE, COL)
        WRITE (6, 510)
510     FORMAT (//' Do you wish to limit the types of records in
        1 the report? '/' (For example, records with PRICE GT 20000'/
        2' or records with DEPARTMENT EQ "SERVICE" SORTED BY
        3 EMPLOYEE_NUMBER)' /' Enter YES or NO: ', $)
        READ (5, 1) ANSWER
1       FORMAT(A)

C If the user wants a record selection expression on his report,
C prompt the user for it.

        IF ((ANSWER(1:1) .EQ. 'Y') .OR. (ANSWER(1:1) .EQ. 'y')) THEN

C Show the user the previously stored fields.

           DO 515 J=1,I
                WRITE (6, 512) SHOFIELDS(J)
512             FORMAT (1X, A75)
515        CONTINUE
```

(continued on next page)

```
C Allow user to enter a record selection expression and
C pass it to DATATRIEVE.

            WRITE (6, 520)
520         FORMAT (//' Enter an expression such as PRICE GT 2000 or'/
        2 ' DATE_EMPLOYED AFTER "01-JULY-1980")')
            READ (5, 521) CHLINE
521         FORMAT (A)
            CALL DTR$COMMAND (DAB, 'REPORT CURRENT WITH !CMD;', CHLINE)
         ELSE

C Invoke the Report Writer for the whole file.

            CALL DTR$COMMAND (DAB, 'REPORT CURRENT;')
         END IF

C Check for errors.

         IF (DAB$W_STATE  .EQ.  DTR$K_STL_MSG) THEN
             GO TO 550
         END IF

C Prompt the user for a report title.

525      WRITE (6, 530)
530      FORMAT (//' Enter the report title enclosed in quotation marks'
         1/' Separate lines with a slash (/)'/
         2' For example, enter:'/'      "RATES SCHEDULE"/"DOMESTIC"'
         3/ 'to produce the title:      RATES SCHEDULE'/
         4  '                               DOMESTIC')
         READ (5, 532) LGTH, RPTHDR
532      FORMAT (Q, A)

C Set the report title.

         IF (LGTH .EQ. 0) THEN
            CALL DTR$COMMAND (DAB, 'SET REPORT_NAME="";')
         ELSE
            CALL DTR$COMMAND (DAB, 'SET REPORT_NAME=!CMD;', RPTHDR)
         END IF
         IF (DAB$W_STATE  .EQ.  DTR$K_STL_MSG) THEN
            GO TO 550
         END IF

C Set a few more characteristics of the report.

         CALL DTR$COMMAND (DAB, 'SET LINES_PAGE = 22')
         CALL DTR$COMMAND (DAB, 'AT BOTTOM OF PAGE PRINT SKIP 1, COL 1,
         1")))))PRESS (RET) TO CONTINUE OR ENTER ""EXIT"" TO STOP: ";')

C Show the user the previously stored fields.

         DO 538 J=1,I
                WRITE (6, 107) SHOFIELDS(J)
107             FORMAT (1X, A75)
538      CONTINUE
```

(continued on next page)

```
C Prompt the user for field names.

        WRITE (6, 540)
540     FORMAT (' Enter the fields you wish to have in the rep
        1ort. Separate them by commas. ')
        READ (5, 1) CHLINE

C Pass the field names to DATATRIEVE.

        CALL DTR$COMMAND (DAB, 'PRINT !CMD;', CHLINE)
        CALL DTR$COMMAND (DAB, 'END_REPORT;')

        CALL LIB$ERASE_PAGE (LINE, COL)
        CALL DTR$CONTINUE (DAB)

C Print out the report.

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

C Check to see if the current line is the bottom of the page.

           IF (MSG_BUFF(1:5) .EQ. '>>>>>') THEN
                WRITE (6, 545) MSG_BUFF
545             FORMAT (1X, A<DAB$W_MSG_LEN>, $)

C If user wishes to stop, cancel the remainder of the report.

                READ (5, 1, END = 546)CR
                IF ((CR .EQ. 'EXIT') .OR. (CR .EQ. 'exit')) THEN
546                CALL DTR$CONTINUE (DAB)
                   CALL DTR$UNWIND (DAB)
                   CALL DTR$CONTINUE (DAB)
                   CALL DTR$DTR (DAB, DTR$M_OPT_CMD)
                   RETURN
                END IF

C Else clear the screen and print the next page.

                CALL LIB$ERASE_PAGE (LINE, COL)
                CALL DTR$CONTINUE (DAB)
                CALL DTR$CONTINUE (DAB)
           END IF
           IF (DAB$W_STATE .EQ. DTR$K_STL_LINE) THEN
                WRITE (6, 547) MSG_BUFF(1:DAB$W_MSG_LEN)
547             FORMAT (1X, A)
                CALL DTR$CONTINUE (DAB)
           END IF
           IF (DAB$L_CONDITION .NE. %LOC(DTR$_SUCCESS)) THEN
               GOTO 550
           END IF
        END DO

550     IF (DAB$L_CONDITION .EQ. %LOC(DTR$_SUCCESS)) THEN
```

```
C Call the terminal server to handle message at the end of the report.

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

C If not successful, display the error message, unwind the
C report and prompt the user to start over

        ELSE
            WRITE (6, 547) MSG_BUFF(1:DAB$W_MSG_LEN)
            CALL DTR$CONTINUE (DAB)
            CALL DTR$UNWIND (DAB)
            CALL DTR$CONTINUE (DAB)
            CALL DTR$DTR (DAB, DTR$M_OPT_CMD)
            WRITE (6, 560)
560         FORMAT (' An error was found by the
        1Report Writer,'/' >>>>>Do you want to try again? ', $)
            READ (5, 1) ANSWER
            IF ((ANSWER(1:1) .EQ. 'Y') .OR.
        1       (ANSWER(1:1) .EQ. 'y')) THEN
                    GO TO 505
            END IF
        END IF
        RETURN
        END
```

3.1.7 The Subroutine STORE

```
C**********************************************************
C                    SUBROUTINE STORE                     *
C Enables the user to store records in the current domain. *
C**********************************************************

        SUBROUTINE STORE (DOMAIN)
        INCLUDE 'DTR$LIBRARY:DAB'

        CHARACTER*31 DOMAIN
        INTEGER*4    NUMBER, DTR_OPTIONS
        INTEGER*2    LINE/1/, COL/1/

C Select DTR$DTR options.

            DTR_OPTIONS =
        1         DTR$M_OPT_CMD           ! Return on DTR$K_STL_CMD
        2       + DTR$M_OPT_CONTROL_C     ! Enable Control C handling

C Erase the screen.

        CALL LIB$ERASE_PAGE (LINE, COL)

C Prompt the user for the number of records to be stored.

        WRITE (6, 630)
630     FORMAT (' Enter the number of records you wish to store: ', $)
        READ (5,640)NUMBER
640     FORMAT (I4)

        CALL LIB$ERASE_PAGE (LINE, COL)

        CALL DTR$COMMAND (DAB, 'REPEAT !VAL STORE !CMD;',%DESCR(NUMBER),
        1                      DOMAIN)

C Call the terminal server to prompt for the values.

        CALL DTR$DTR (DAB, DTR_OPTIONS)

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

        CALL DTR$COMMAND (DAB,'FIND !CMD;',DOMAIN)
        CALL MESSAGE

        RETURN
        END
```

3.1.8 The Subroutine CHOOSE

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

        SUBROUTINE CHOOSE (DOMAIN)
        INCLUDE 'DTR$LIBRARY:DAB'
        CHARACTER*31 DOMAIN
        LOGICAL      NO_DOMAIN/.TRUE./

        DO WHILE (NO_DOMAIN)
            CALL DTR$COMMAND (DAB, 'SHOW DOMAINS;')

C Use DTR$DTR to display the result of SHOW DOMAINS.

            CALL DTR$DTR (DAB, DTR$M_OPT_CMD)

C Ask the user for the domain and ready it.

            WRITE (6, 20)
20          FORMAT (' Enter the name of the domain you want to use: ',$)
            READ (5, 1) DOMAIN
1           FORMAT (A)
                CALL DTR$COMMAND (DAB, 'READY !CMD WRITE;', DOMAIN)

C Check for an error in readying the domain; reprompt if there was.
C Form a collection of all records in the domain and check for errors.

                CALL DTR$DTR (DAB, DTR$M_OPT_CMD)

                IF (DAB$L_CONDITION .NE. %LOC(DTR$_SUCCESS)) THEN
                    WRITE (6, 40)
40                  FORMAT (' Try again....')
                ELSE
                    NO_DOMAIN = .FALSE.
                    CALL DTR$COMMAND (DAB, 'FIND !CMD;', DOMAIN)
                    CALL MESSAGE
                    IF (DAB$L_CONDITION .NE. %LOC(DTR$_SUCCESS)) THEN
                        NO_DOMAIN = .TRUE.
                    END IF
                END IF
        END DO
        NO_DOMAIN = .TRUE.

        RETURN
        END
```

3.2 Creating a User-Defined Keyword

The program CORRELATE adds the user-defined keyword CORRELATE to interactive DATATRIEVE. Users who run the program can use DATATRIEVE in the usual manner. In addition, they can use the statement CORRELATE.

The format of this statement is:

CORRELATE independent-field [,] dependent-field [OF rse]

The following examples show how you can use CORRELATE:

```
DTR> READY ANNUAL_REPORT
DTR> CORRELATE RESEARCH, NET_INCOME OF ANNUAL_REPORT


       LINEAR
      REGRESSION
         LINE

 NET_INCOME =            -11.5668 +              1.3894 * RESEARCH


      COEFFICIENT
          OF
      CORRELATION

       0.99749

DTR> FIND ANNUAL_REPORT WITH DATE BEFORE "01-JAN-1978"
[7 records found]
DTR> BEGIN
CON>    CORRELATE EMPLOYEES, SERVICES
CON>    CORRELATE EMPLOYEES, NET_INCOME_PER_SHARE
CON> END


       LINEAR
      REGRESSION
         LINE

 SERVICES =            -112.9101 +              0.0132 * EMPLOYEES


      COEFFICIENT
          OF
      CORRELATION

       0.80055

       LINEAR
      REGRESSION
         LINE
 NET_INCOME_PER_SHARE =       -1.1342 +          0.0002 * EMPLOYEES


      COEFFICIENT
          OF
      CORRELATION

       0.82269

DTR>
```

One program and two subroutines create the UDK CORRELATE:

- The main program, CORRELATE.FOR, initializes DATATRIEVE and invokes the terminal server. The program simulates interactive DATATRIEVE until the user enters CORRELATE. The program then calls PARSE.
- The subroutine PARSE:
  - Parses the CORRELATE statement.
  - Converts each CORRELATE statement the user entered into one DATATRIEVE BEGIN-END statement. For example, if the user enters the statement:

    ```
    CORRELATE RESEARCH, NET_INCOME OF ANNUAL_REPORT
    ```

    The subroutine PARSE converts it into the following BEGIN-END statement:

    ```
    BEGIN
      DECLARE PORT PT1 USING
        01 NUM PIC 9(4) COMP.
      DECLARE PORT PT2 USING
        01 WHOLE.
           02 PART_A REAL.
           02 PART_B REAL.
      STORE PT1 USING NUM = COUNT OF ANNUAL_REPORT
      STORE PT2 USING
        BEGIN
          PART_A = TOTAL RESEARCH OF ANNUAL_REPORT
          PART_B = TOTAL NET_INCOME OF ANNUAL_REPORT
        END
      FOR ANNUAL_REPORT
        BEGIN
          STORE PT2 USING
            BEGIN
              PART_A = RESEARCH
              PART_B = NET_INCOME
            END
         END
       DISPLAY RESEARCH
       DISPLAY NET_INCOME
    END
    ```

  - Passes the BEGIN-END statement to DATATRIEVE.

  - Returns control to the main program.

- The subroutine CALCULATE uses the values passed in the ports to calculate the linear regression and coefficient of correlation. It then displays the results.

The main program and the subroutines include a file called COMMON.FOR. This file contains variable declarations common to all three. Here is the file COMMON.FOR:

```
CHARACTER*31 FIELD1
CHARACTER*1  COMMA
CHARACTER*31 FIELD2
CHARACTER*2  WORD_OF
CHARACTER*80 EXPRESSION
INTEGER*4    INIT_OPTS
COMMON FIELD1, FIELD2, EXPRESSION, INIT_OPTS
```

3.2.1 The Main Program: CORRELATE

```
C
C                    C O R R E L A T E . F O R
C
C Include the DAB and the common variables.
C
        INCLUDE  'DTR$LIBRARY:DAB'
        INCLUDE  'DTR$LIBRARY:COMMON'
        INTEGER*4 DTR$DTR
        INTEGER*4 DTR$GET_STRING
        INTEGER*2 DTR_OPTS
        INTEGER   RET_STATUS
C
C Declare the normal and exit status.
C
        EXTERNAL SS$_NORMAL
        EXTERNAL DTR$_EXIT
C
C Select options and initialize the session with DATATRIEVE.
C
            INIT_OPTS =
        1       + DTR$K_SEMI_COLON_OPT
        2       + DTR$K_UNQUOTED_LIT
        3       + DTR$K_FORMS_ENABLE
        4       + DTR$K_SYNTAX_PROMPT

        CALL DTR$INIT (DAB, 100, MSG_BUFF, AUX_BUFF, INIT_OPTS)
C
C Use the DTR$CREATE_UDK call to create the UDKs CORRELATE and SPAWN.
C
        CALL DTR$CREATE_UDK (DAB, 'CORRELATE', 1,DTR$K_UDK_STATEMENT)
        CALL DTR$CREATE_UDK (DAB, 'SPAWN', 2, DTR$K_UDK_COMMAND)
C
C Declare the DTR$DTR call options.
C
        DTR_OPTS =
        1       + DTR$M_OPT_PGET       ! Return on DTR$K_STL_PGET
        2       + DTR$M_OPT_CONTROL_C  ! Enable Control C handling
        3       + DTR$M_OPT_STARTUP    ! Execute startup command file
        4       + DTR$M_OPT_FOREIGN    ! Execute invocation command lines
        5       + DTR$M_OPT_BANNER     ! Display DTR banner
        6       + DTR$M_OPT_UDK        ! Return on DTR$K_STL_UDK
        7       + DTR$M_OPT_END_UDK    ! Return on DTR$K_STL_END_UDK
C
C Continue until the user has entered EXIT or CTRL/Z at the
C DTR> prompt.
C
        DO WHILE (RET_STATUS .NE. %LOC(DTR$_EXIT))
C
C Call the DATATRIEVE terminal server.
C
          RET_STATUS = DTR$DTR (DAB, DTR_OPTS)
```

(continued on next page)

```
C
C If the terminal server call returns a status other than success or
C exit, then stop the program.
C
          IF ((RET_STATUS .NE. %LOC(SS$_NORMAL)) .AND.
        1     (RET_STATUS .NE. %LOC(DTR$_EXIT))) THEN
                WRITE (6, *) 'Error in calling terminal server.'
                STOP
          END IF
C
C If the user enters a UDK, the program gets control at this point.
C If the UDK entered is CORRELATE, then call PARSE.
C
C The PARSE subroutine retrieves the rse the user entered and parses
C it. The subroutine then passes a BEGIN-END statement to DATATRIEVE
C in place of the arguments the user entered.
C
C DATATRIEVE checks the statement for errors and executes it.
C
          IF (DAB$W_STATE .EQ. DTR$K_STL_UDK) THEN
             IF (DAB$W_UDK_INDEX .EQ. 1) THEN
                CALL PARSE()
             END IF
C
C Call a Run-Time Library routine to spawn a subprocess
C if the user entered SPAWN.
C
             IF (DAB$W_UDK_INDEX .EQ. 2) THEN
                CALL LIB$SPAWN()
             END IF
C
C Terminate the UDK and continue.
C
             CALL DTR$END_UDK (DAB)
          END IF
C
C DATATRIEVE returns to the program with the DTR$K_STL_PGET
C stallpoint.
C
C Call the subroutine CALCULATE to handle the DATATRIEVE commands
C issued by PARSE.FOR.
C
          IF (DAB$W_STATE .EQ. DTR$K_STL_PGET) THEN
            CALL CALCULATE()
          END IF

        END DO
C
C If EXIT or CTRL/Z is entered, end the session.
C
999     CALL DTR$FINISH (DAB)

        END
```

3.2.2 The Subroutine PARSE

```
C*****************************************************************
C                    SUBROUTINE PARSE                            *
C Parses the arguments to the CORRELATE statement, converts the  *
C statement into a BEGIN-END block, and passes the block to      *
C DATATRIEVE for processing.                                     *
C*****************************************************************

C
C   Declare variables and include definition of the DAB.
C
        SUBROUTINE   PARSE()

        INCLUDE      'DTR$LIBRARY:DAB'
        INCLUDE      'DTR$LIBRARY:COMMON'
        CHARACTER*33 FLD1
        CHARACTER*33 FLD2
        INTEGER*4    DTR$GET_STRING
        INTEGER      RET_STATUS
        INTEGER      COMMA_LENGTH
        INTEGER      OF_LENGTH
        INTEGER      NUM_TOKENS
C
C Declare the status messages.
C
        EXTERNAL  DTR$_ENDOFSTR
        EXTERNAL  DTR$_MORESTR
        EXTERNAL  SS$_NORMAL
C
C  Set the variable NUM_TOKENS to the number of tokens needed (3).
C  Set token length variables for the test_token.
C
        NUM_TOKENS = 3
        COMMA_LENGTH = 1
        OF_LENGTH = 2
C
C Use the DTR$GET_STRING call to get the first token. This should be
C the name of the independent field.
C
C If there was no token, DATATRIEVE returns the status (DTR$_ENDOFSTR).
C If this status was returned, set the number of tokens received to 0.
C
        RET_STATUS = DTR$GET_STRING (DAB, DTR$K_TOK_TOKEN,
        1       FIELD1)
        IF (RET_STATUS .EQ. %LOC(DTR$_ENDOFSTR)) THEN
                NUM_TOKENS = 0
        END IF
```

(continued on next page)

```
C
C If there was a token for FIELD1, DATATRIEVE returns the
C status DTR$_MORESTR. If this was returned, then make another
C DTR$GET_STRING call to DATATRIEVE.
C
C The next token may be the optional comma.
C Make the call using DTR$K_TOK_TEST_TOKEN.
C
C If the token length is 1 and the token is a comma,
C DATATRIEVE places the token in the field named COMMA.
C Otherwise, DATATRIEVE returns a status of DTR$_MORESTR.
C
C If there was no token, DATATRIEVE returns DTR$_ENDOFSTR.
C In this case, set the number of tokens to 1.
C
        IF (RET_STATUS .EQ. %LOC(DTR$_MORESTR)) THEN
            RET_STATUS = DTR$GET_STRING( DAB, DTR$K_TOK_TEST_TOKEN,
        1      COMMA, COMMA_LENGTH, ',')
             IF (RET_STATUS .EQ. %LOC(DTR$_ENDOFSTR) )THEN
                NUM_TOKENS = 1
             END IF
        END IF
C
C If DTR$_MORESTR was returned, then make another DTR$GET_STRING call.
C The next token should be the dependent field (FIELD2).
C
C If the status is DTR$_ENDOFSTR, then there was no token for FIELD2.
C Set the number of tokens to 1. (The number of tokens does not
C include the comma or the word OF).
C
        IF (RET_STATUS .EQ. %LOC(DTR$_MORESTR)) THEN
             RET_STATUS = DTR$GET_STRING( DAB, DTR$K_TOK_TOKEN,
        1      FIELD2 )
             IF (RET_STATUS .EQ. %LOC(DTR$_ENDOFSTR) )THEN
                NUM_TOKENS = 1
             END IF
        END IF
C
C If the status is DTR$_MORESTR, make another DTR$GET_STRING
C call. The next token may be the optional word OF. Make
C the call using DTR$K_TOK_TEST_TOKEN.
C
C If the token length is 2 and the token is OF, DATATRIEVE places
C the token in the field named WORD_OF; otherwise, it returns the
C status DTR$_MORESTR.
C
C If there was no token, DATATRIEVE returns DTR$_ENDOFSTR.
C In this case, the program sets the number of tokens
C to 3 and sets the last token to CURRENT.
C CORRELATE then uses the CURRENT collection.
C
        IF (RET_STATUS .EQ. %LOC(DTR$_MORESTR)) THEN
            RET_STATUS = DTR$GET_STRING( dab, DTR$K_TOK_TEST_TOKEN,
        1      WORD_OF, OF_LENGTH, 'OF')
            IF ((RET_STATUS .EQ. %LOC(DTR$_ENDOFSTR)) .OR.
        2        (OF_LENGTH .EQ. 0) )THEN
                EXPRESSION = 'CURRENT'
                NUM_TOKENS = 3
            END IF
        END IF
```

```
C
C If DTR$_MORESTR was returned, make another DTR$GET_STRING
C call.
C
C The rest of the line the user entered should be a record
C selection expression. Use the DTR$K_TOK_COMMAND token type
C to place the remaining tokens in EXPRESSION.
C
C If the status is DTR$_ENDOFSTR, no RSE was entered. Set the
C number of tokens to 2.
C
        IF (RET_STATUS .EQ. %LOC(DTR$_MORESTR)) THEN
            RET_STATUS = DTR$GET_STRING( dab, DTR$K_TOK_COMMAND,
        1       EXPRESSION )
            IF (RET_STATUS .EQ. %LOC(DTR$_ENDOFSTR) )THEN
                NUM_TOKENS = 2
            END IF
        END IF
C
C   (The user entered DTR> CORRELATE <RET>)
C
C If the number of tokens is 0, then prompt the user for the
C name of an independent field.
C
        IF (NUM_TOKENS .LT. 1) THEN
            WRITE (6, *)'ENTER INDEPENDENT FIELD: '
            READ (5, 110) FIELD1
110     FORMAT(A)
        END IF
C
C   (The user entered DTR> CORRELATE field1<RET> or
C                     DTR> CORRELATE field1,<RET>)
C
C If the number of tokens is 0 or 1, then prompt the user for
C the name of a dependent field.
C
        IF (NUM_TOKENS .LT. 2) THEN
            WRITE (6, *)'ENTER DEPENDENT FIELD: '
            READ (5, 110) FIELD2
        END IF
C
C   (The user entered DTR> CORRELATE field1 field2 OF<RET>)
C
C If the number of tokens is 0, 1, or 2, then prompt the user
C for an RSE.
C
        IF (NUM_TOKENS .LT. 3) THEN
            WRITE (6, *)'ENTER A RECORD SELECTION EXPRESSION: '
            READ (5, 110) EXPRESSION
            IF (INDEX(EXPRESSION,'    ') .EQ. 1) THEN
                EXPRESSION = 'CURRENT'
            END IF
        END IF
```

(continued on next page)

```
C
C Use a library routine to find the first blank space in the variables
C for the field names. Because field names cannot contain blanks,
C this will return the length of the field name along with one space.
C
C Store the field names with quotation marks around them into the
C temporary storage variables, FLD1 and FLD2.
C
200     LEN = LIB$LOCC(' ',FIELD1)
        FLD1 = '"'//FIELD1(1:LEN)//'"'
        LEN = LIB$LOCC(' ',FIELD2)
        FLD2 = '"'//FIELD2(1:LEN)//'"'
C
C Change the INIT_OPTIONS to just DTR$K_MORE_COMMANDS.
C This allows the program to pass all of the command lines
C before DATATRIEVE starts checking for syntax errors.
C
        DAB$L_OPTIONS = DTR$K_MORE_COMMANDS
C
C Pass DATATRIEVE all of the statements required to
C retrieve and process the data for a linear regression
C equation and the coefficient of correlation.
C
C All statements should be part of a BEGIN-END block.
C
        CALL DTR$COMMAND (DAB, 'BEGIN')
C
C Declare the ports that will be needed.  These ports are
C deleted at the end of the BEGIN-END statement.
C
C Port PT1 is for the number of records to be used.
C Port PT2 is for the total of the fields in the collection
C and for the individual values per record.
C
        CALL DTR$COMMAND (DAB, 'DECLARE PORT PT1 USING ')
        CALL DTR$COMMAND (DAB, '    01 NUM PIC 9(4) COMP.;')
        CALL DTR$COMMAND (DAB, 'DECLARE PORT PT2 USING ')
        CALL DTR$COMMAND (DAB, '    01 WHOLE.')
        CALL DTR$COMMAND (DAB, '       02 PART_A REAL.')
        CALL DTR$COMMAND (DAB, '       02 PART_B REAL.;')
C
C Store the total number of records in the RSE into port PT1.
C
        CALL DTR$COMMAND (DAB,
        1       'STORE PT1 USING NUM = COUNT OF !CMD;', EXPRESSION)
C
C Store the total of both fields in the RSE into port PT2.
C
        CALL DTR$COMMAND (DAB, 'STORE PT2 USING')
        CALL DTR$COMMAND (DAB, '    BEGIN')
        CALL DTR$COMMAND (DAB, '    PART_A= TOTAL !CMD OF !CMD;',
        1       FIELD1, EXPRESSION)
        CALL DTR$COMMAND (DAB, '    PART_B= TOTAL !CMD OF !CMD;',
        1       FIELD2, EXPRESSION)
        CALL DTR$COMMAND (DAB, '    END;')
```

```
C
C One at a time, store the value of both fields for each record
C into port PT2
C
        CALL DTR$COMMAND (DAB, 'FOR !CMD',EXPRESSION)
        CALL DTR$COMMAND (DAB, '  BEGIN ')
        CALL DTR$COMMAND (DAB, '    STORE PT2 USING ')
        CALL DTR$COMMAND (DAB, '        BEGIN')
        CALL DTR$COMMAND (DAB, '        PART_A= !CMD;', FIELD1)
        CALL DTR$COMMAND (DAB, '        PART_B= !CMD;', FIELD2)
        CALL DTR$COMMAND (DAB, '        END;')
        CALL DTR$COMMAND (DAB, '  END')
C
C After all records in the RSE have been stored into PT2,
C instruct DTR to display the names of both fields used.
C This serves two purposes:
C
C 1. The subroutine CALCULATE knows when DATATRIEVE has finished
C    storing the records in the RSE.
C
C 2. DATATRIEVE passes the names of the fields to CALCULATE in the
C    auxiliary buffer, and the length of the names in DAB$W_AUX_LEN.
C    The field names are saved if more than one CORRELATE
C    statement is entered within one statement.
C
        CALL DTR$COMMAND (DAB, 'DISPLAY !CMD;', FLD1)
        CALL DTR$COMMAND (DAB, 'DISPLAY !CMD;', FLD2)
C
C Change the options back to the INIT_OPTIONS declared in the INIT call.
C
        DAB$L_OPTIONS = INIT_OPTS
C
C End the BEGIN_END block.
C
        CALL DTR$COMMAND (DAB, 'END;')
C
C Return to the main program.
C
        RETURN
        END
```

3.2.3 The Subroutine CALCULATE

```
C*****************************************************************
C                     SUBROUTINE CALCULATE                       *
C Performs the calculations and prints out the linear regression  *
C equation and coefficient of correlation.                        *
C*****************************************************************

        SUBROUTINE CALCULATE()

C***************************************************************
C
C       Formulas used to find the linear equation:
C
C       LINEAR EQUATION :  Y = bX + a
C
C       Equation to arrive at value for b:
C                       (note:  E = summation
C                               n = number of data elements used)
C
C                b = E(X*Y) - n(average(X) * average(Y))
C                    ----------------------------------
C                    E(X**2) - n(average(x)**2)
C
C       Equation to arrive at value for a:
C
C               a = average(Y) - (b * average(X))
C
C       Variables used:
C
C               E(X*Y) = SUMXY
C               E(X**2) = SUMX2
C               N = COUNTER
C
C***************************************************************
C
C       Formulas used to find the coefficient of correlation:
C
C                         a*E(Y) + b*E(X*Y) - n*(average(Y)**2)
C       CO_OF_CORR_SQRD = ------------------------------------
C                               E(Y**2) - n*(average(Y)**2)
C
C       COEFFICIENT OF CORRELATION:
C
C       CO_OF_CORR = square root of (CO_OF_CORR_SQRD)
C
C       Variables used:
C               E(Y) = SUMY
C               E(X*Y) = SUMXY
C               E(Y**2) = SUMY2
C               N = COUNTER
C
C***************************************************************
C
```

```
C Include DAB and common variables.
C
        INCLUDE 'DTR$LIBRARY:DAB'
        INCLUDE 'DTR$LIBRARY:COMMON'
        CHARACTER*80 OUTPUT_LINE
        INTEGER*4 COUNTER/0/
        INTEGER*4 DTR_OPTS
        INTEGER*4 LEN1
        INTEGER*4 LEN2
        REAL*4 REC_BUFF(2)
        REAL*8 AVE(2)
        REAL*8 SUMY
        REAL*8 SUMXY
        REAL*8 SUMX2
        REAL*8 SUMY2
        REAL*8 TOP_LIN
        REAL*8 BOTTOM_LIN
        REAL*8 TOP_CORR
        REAL*8 BOTTOM_CORR
        REAL*8 CO_OF_CORR_SQRD
        REAL*8 CO_OF_CORR
C
C Declare status SS$_NORMAL and the error message DTR$_UNWIND as
C external values.
C
        EXTERNAL SS$_NORMAL
        EXTERNAL DTR$_UNWIND
C
C Declare the default format for output.
C
100     FORMAT (/1x,A)
C
C Select DTR$DTR options.
C
            DTR_OPTS =
        1         DTR$M_OPT_CMD    ! Return on DTR$K_STL_CMD
        2       + DTR$M_OPT_PGET   ! Return on DTR$K_STL_PGET
C
C Three STORE port-name statements were passed to DATATRIEVE
C in the BEGIN-END block in the subroutine PARSE. The first
C stored the count of records into PT1.
C
C If DATATRIEVE is at DTR$K_STL_PGET, retrieve the count
C from PT1.
C
        IF (DAB$W_STATE .EQ. DTR$K_STL_PGET) then
            CALL DTR$GET_PORT (DAB, COUNTER)
        END IF
```

(continued on next page)

```
C
C While DATATRIEVE is at the stallpoint DTR$K_STL_MSG,
C return to the main program if the user enters CTRL/C.
C Otherwise print out the message, write it to any open
C log file, then call DTR$CONTINUE.
C
        DO WHILE (DAB$W_STATE .EQ. DTR$K_STL_MSG)
            IF (DAB$L_CONDITION .EQ. %LOC(DTR$_UNWIND)) THEN
                RETURN
            END IF
            IF (DAB$L_CONDITION .NE. %LOC(DTR$_SUCCESS)) THEN
                    WRITE (6, 100) MSG_BUFF
                    CALL DTR$PUT_OUTPUT (DAB, MSG_BUFF)
            END IF
            CALL DTR$CONTINUE (DAB)
        END DO
C
C Check to see if there are no records that match the RSE entered.
C If there are no records, then there is no linear regression or
C coefficient of correlation.
C
        IF (COUNTER .EQ. 0) THEN
C
C DATATRIEVE is at DTR$K_STL_PGET because of the two STORE PORT
C statements. Call DTR$GET_PORT until DATATRIEVE is at DTR$K_STL_MSG.
C Then call DTR$CONTINUE until DATATRIEVE is at DTR$K_STL_CMD.
C
C Display error messages on the screen and to a file the user may have
C opened.
C
                DO WHILE (DAB$W_STATE .EQ. DTR$K_STL_PGET)
                    CALL DTR$GET_PORT (DAB, REC_BUFF)
                END DO
                DO WHILE (DAB$W_STATE .EQ. DTR$K_STL_MSG)
                    CALL DTR$CONTINUE (DAB)
                END DO
                WRITE (6, *)
        1       '0 Records Found, No valid regression line'
                CALL DTR$PUT_OUTPUT (DAB,
        1       '0 Records Found, No valid regression line')
                RETURN
        END IF
C
C Check to see if there was only 1 record found.  If there is
C only 1 record, then there is no linear regression or
C coefficient of correlation.
C
        IF (COUNTER .EQ. 1) THEN
                DO WHILE (DAB$W_STATE .EQ. DTR$K_STL_PGET)
                    CALL DTR$GET_PORT (DAB, REC_BUFF)
                END DO
                DO WHILE (DAB$W_STATE .EQ. DTR$K_STL_MSG)
                    CALL DTR$CONTINUE (DAB)
                END DO
                WRITE (6, *)
        1       '1 Record Found, No valid regression line'
                CALL DTR$PUT_OUTPUT (DAB,
        1       '1 Record Found, No valid regression line')
                RETURN
        END IF
```

```
C
C Initialize the summation variables.
C
        SUMXY=0
        SUMX2=0
        SUMY2=0
C
C If DATATRIEVE is at DTR$K_STL_PGET, it has stored the totals of the
C fields into PT2. Get the totals in the port and pass them to a
C variable in the program.
C
        IF (DAB$W_STATE .EQ. DTR$K_STL_PGET) then
            CALL DTR$GET_PORT (DAB, REC_BUFF)
        END IF
C
C While DATATRIEVE is at the stallpoint DTR$K_STL_MSG,
C return to the main program if the user enters CTRL/C.
C Otherwise print out the message, write it to any open
C log file, then call DTR$CONTINUE.
C
        DO WHILE (DAB$W_STATE .EQ. DTR$K_STL_MSG)
            IF (DAB$L_CONDITION .EQ. %LOC(DTR$_UNWIND)) THEN
                RETURN
            END IF
            IF (DAB$L_CONDITION .NE. %LOC(DTR$_SUCCESS)) THEN
                    WRITE (6, 100) MSG_BUFF
                    CALL DTR$PUT_OUTPUT (DAB, MSG_BUFF)
            END IF
            CALL DTR$CONTINUE (DAB)
        END DO
C
C Move the total of FIELD2 into the variable for the summation of y.
C
        SUMY = REC_BUFF(2)
C
C Find the average values of the two fields.
C
136     AVE(1) = REC_BUFF(1) / COUNTER
        AVE(2) = REC_BUFF(2) / COUNTER
C
C If DATATRIEVE is at DTR$K_STL_PGET, it has stored the values for
C individual fields in the record stream.
C
140     DO WHILE (DAB$W_STATE .EQ. DTR$K_STL_PGET)
C
C Get the field value and pass it to a variable in the program.
C
            CALL DTR$GET_PORT (DAB, REC_BUFF)
C
C Compute the sum of the two variables after multiplying
C them. Compute the sum of the squares of FIELD1 and
C FIELD2.
C
            SUMXY = SUMXY + (REC_BUFF(1)*REC_BUFF(2))
            SUMX2 = SUMX2 + (REC_BUFF(1)**2)
            SUMY2 = SUMY2 + (REC_BUFF(2)**2)
        END DO
```

(continued on next page)

```
C
C If the user entered CTRL/C, return to the main program.
C
        IF ((DAB$W_STATE .EQ. DTR$K_STL_MSG) .AND.
        1   (DAB$L_CONDITION .EQ. %LOC(DTR$_UNWIND))) THEN
                RETURN
        END IF
C
C Finish computation of the linear regression line.
C
        TOP_LIN = (SUMXY - (COUNTER*AVE(1)*AVE(2)))
        BOTTOM_LIN = (SUMX2 - (COUNTER*(AVE(1)**2)))
C
C If all values of FIELD1 are the same, then there is no regression
C line. Display a message to the user.
C
        IF (BOTTOM_LIN .EQ. 0) THEN
                WRITE (6, *)'There is no valid linear regression line'
                CALL DTR$PUT_OUTPUT (DAB,
        1       'There is no valid linear regression line')
                DO WHILE (DAB$W_STATE .EQ. DTR$K_STL_MSG)
                    CALL DTR$CONTINUE (DAB)
                END DO
                RETURN
        END IF
        B = TOP_LIN / BOTTOM_LIN
        A = AVE(2) - (B * AVE(1) )
C
C Finish computation of coefficient of correlation.
C
        TOP_CORR = ((A*SUMY) + (B*SUMXY) - (COUNTER*(AVE(2)**2)))
        BOTTOM_CORR = (SUMY2 - (COUNTER*(AVE(2)**2)))
C
C If all values of FIELD2 are the same, then there is no coefficient of
C correlation. Display a message to the user.
C
        IF (BOTTOM_CORR .EQ. 0) THEN
                WRITE (6, *)'There is no valid coefficient
        1of correlation'
                CALL DTR$PUT_OUTPUT (DAB,
        1       'There is no valid coefficient of correlation')
                DO WHILE (DAB$W_STATE .EQ. DTR$K_STL_MSG)
                    CALL DTR$CONTINUE (DAB)
                END DO
                RETURN
        END IF
        CO_OF_CORR_SQRD = TOP_CORR / BOTTOM_CORR
        CO_OF_CORR = DSQRT (CO_OF_CORR_SQRD)
C
C If the user entered CTRL/C, return to the main program.
C
        IF ((DAB$W_STATE .EQ. DTR$K_STL_MSG) .AND.
        1   (DAB$L_CONDITION .EQ. %LOC(DTR$_UNWIND))) THEN
                RETURN
        END IF
```

```
C
C Retrieve the name of the first field from the auxiliary buffer.
C Save the length of the field name in another variable.
C
        FIELD1 = AUX_BUFF
        LEN1 = DAB$W_AUX_LEN
        CALL DTR$CONTINUE (DAB)
C
C If the user entered CTRL/C, return to the main program.
C
        IF ((DAB$W_STATE .EQ. DTR$K_STL_MSG) .AND.
        1   (DAB$L_CONDITION .EQ. %LOC(DTR$_UNWIND))) THEN
                RETURN
        END IF
C
C Retrieve the name of the second field from the auxiliary buffer.
C Save the length of the field name in another variable.
C
        FIELD2 = AUX_BUFF
        LEN2 = DAB$W_AUX_LEN
        CALL DTR$CONTINUE (DAB)
C
C If the user entered CTRL/C, return to the main program.
C
        IF ((DAB$W_STATE .EQ. DTR$K_STL_MSG) .AND.
        1   (DAB$L_CONDITION .EQ. %LOC(DTR$_UNWIND))) THEN
                RETURN
        END IF
C
C Print headers for the linear regression line.
C
        WRITE (6, 220)
220     FORMAT (//7X, 'LINEAR'/5X, 'REGRESSION'/8X, 'LINE'/)
C
C Print headers for the linear regression line to a file if one
C was opened.
C
        CALL DTR$PUT_OUTPUT (DAB, '  ')
        CALL DTR$PUT_OUTPUT (DAB, '  ')
        CALL DTR$PUT_OUTPUT (DAB, '        LINEAR')
        CALL DTR$PUT_OUTPUT (DAB, '      REGRESSION')
        CALL DTR$PUT_OUTPUT (DAB, '         LINE')
        CALL DTR$PUT_OUTPUT (DAB, '  ')
C
C Print the linear regression line.
C
        WRITE (6, 225) FIELD2 (1:LEN2), A, B, FIELD1 (1:LEN1)
225     FORMAT (1X, A, '= ', F20.4, ' + ', F20.4, ' * ', A)
        WRITE (OUTPUT_LINE, 225) FIELD2 (1:LEN2), A, B, FIELD1 (1:LEN1)
        CALL DTR$PUT_OUTPUT (DAB, OUTPUT_LINE)
```

(continued on next page)

```
C
C Print headers for the coefficient of correlation.
C
        WRITE (6, 230)
230     FORMAT (//5X, 'COEFFICIENT'/9X, 'OF'/5X, 'CORRELATION'/)
        CALL DTR$PUT_OUTPUT (DAB, ' ')
        CALL DTR$PUT_OUTPUT (DAB, ' ')
        CALL DTR$PUT_OUTPUT (DAB, '     COEFFICIENT')
        CALL DTR$PUT_OUTPUT (DAB, '         OF')
        CALL DTR$PUT_OUTPUT (DAB, '     CORRELATION')
        CALL DTR$PUT_OUTPUT (DAB, ' ')
C
C Print the coefficient of correlation.
C
        WRITE (6, 235) CO_OF_CORR
235     FORMAT (7X, F7.5)
        WRITE (OUTPUT_LINE, 235) CO_OF_CORR
        CALL DTR$PUT_OUTPUT (DAB, OUTPUT_LINE)
C
C Leave a blank line after the coefficient of correlation.
C
        WRITE (6, *)' '
        CALL DTR$PUT_OUTPUT (DAB, ' ')
C
C Return to the main program.
C
300     RETURN
        END
```

Sample COBOL Programs 4

This chapter contains several sample COBOL programs that call DATATRIEVE. These programs show how you can call DATATRIEVE to perform calculations on data, to store and retrieve data, and to help end users perform information management tasks. Copies of the programs are in the DTR$LIBRARY directory.

To run these programs, you must compile them and then link them with the DATATRIEVE shareable image:

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

See Chapter 2 for more information on using an options file to link programs that call DATATRIEVE.

4.1 Creating an End-User Interface to DATATRIEVE

The program ENTRY accepts data entered by a user from the terminal and stores the data in the DATATRIEVE domain PERSONNEL. The program uses Run-Time Library procedures to format the terminal screen.

The program uses a port called PERSONNEL_PORT to pass records to DATATRIEVE. Because data is entered from the terminal in ASCII format, all fields in the port are declared as character data types. Here are the DATATRIEVE commands to define the port PERSONNEL_PORT:

```
DEFINE PORT PERSONNEL_PORT USING PERSONNEL_ENTRY_REC;

DEFINE RECORD PERSONNEL_ENTRY_REC USING
01  PERSON.
    05 ID                    PIC  X(5).
    05 EMPLOYEE_STATUS       PIC  X(11).
    05 EMPLOYEE_NAME.
        10 FIRST_NAME        PIC  X(10).
        10 LAST_NAME         PIC  X(10).
    05 DEPT                  PIC  XXX.
    05 START_DATE            PIC  X(11).
    05 SALARY                PIC  X(5).
    05 SUP_ID                PIC  X(5).
;
```

Here is the program ENTRY:

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

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

        COPY "DTR$LIBRARY:DAB.LIB".

*****************************************************
* Set up a record buffer by copying the CDD record  *
* definition.                                       *
*****************************************************

        COPY "PERSONNEL_ENTRY_REC" FROM DICTIONARY.

*****************************************************
* Declare screen formatting variables.              *
*****************************************************
```

```
01 LINE_POS         PIC S9(9) COMP.
01 COL_POS          PIC S9(9) COMP.
01 BOLD_VIDEO       PIC S9(9) COMP VALUE 1.
01 REVERSE_VIDEO    PIC S9(9) COMP VALUE 2.
01 TXT              PIC X(50).
01 SPACE_TXT        PIC X     VALUE " ".
01 HDR_TXT          PIC X(16) VALUE "PERSONNEL RECORD".
01 FNAME_TXT        PIC X(16) VALUE "NAME: __________".
01 LNAME_TXT        PIC X(10) VALUE "__________".
01 STAT_TXT         PIC X(19) VALUE "STATUS: ___________".
01 ID_TXT           PIC X(9)  VALUE "ID: _____".
01 DEPT_TXT         PIC X(15) VALUE "DEPARTMENT: ___".
01 SUP_TXT          PIC X(20) VALUE "SUPERVISOR ID: _____".
01 DATE_TXT         PIC X(24) VALUE "START DATE: ____________".
01 SAL_TXT          PIC X(13) VALUE "SALARY: _____".

01 STACK_SIZE PIC 99 COMP VALUE 100.
01 DTR_OPTIONS PIC 9(9) COMP.
01 CONT PIC X.
01 COMMAND_LINE PIC X(80)
       VALUE "READY PERSONNEL WRITE; READY PERSONNEL_PORT;".

PROCEDURE DIVISION.

010-INITIALIZE-INTERFACE.

**************************************************************
* Initialize the interface with DTR$INIT.  Use DTR$COMMAND to   *
* ready domain and port.                                        *
**************************************************************

        CALL "DTR$INIT" USING DAB STACK-SIZE
                        BY DESCRIPTOR MSG_BUFF AUX_BUFF.

        CALL "DTR$COMMAND" USING DAB
                        BY DESCRIPTOR COMMAND_LINE.

**************************************************************
* Use DTR$DTR to display messages and return control on the     *
* DTR$K_STL_CMD stallpoint.                                     *
**************************************************************

        CALL "DTR$DTR" USING DAB DTR$M_OPT_CMD.

**************************************************************
* Command DATATRIEVE to store PERSONNEL records from the port.  *
* DATATRIEVE stalls at DTR$K_STL_PPUT                           *
**************************************************************

        MOVE SPACES TO COMMAND_LINE.
        MOVE "FOR PERSONNEL_PORT STORE PERSONNEL USING PERSON = PERSON;"
                TO COMMAND_LINE.
        CALL "DTR$COMMAND" USING DAB
                        BY DESCRIPTOR COMMAND_LINE.
```

(continued on next page)

```
100-STORE-A-RECORD.

***************************************************************
* Read in a data record.                                      *
***************************************************************
        PERFORM 200-GET-A-RECORD.

****************************************************************
* Use the call DTR$PUT_PORT to pass the record to DATATRIEVE.  *
****************************************************************

        CALL "DTR$PUT_PORT" USING DAB PERSON.

*******************************************************************
* Use the DTR$DTR call to display messages. Return control to the *
* program on DTR$K_STL_CMD and DTR$K_STL_PPUT.                    *
*******************************************************************

        ADD DTR$M_OPT_CMD DTR$M_OPT_PPUT GIVING DTR_OPTIONS.
        IF DTR$K_STL_MSG THEN
                CALL "DTR$DTR" USING DAB DTR_OPTIONS.
        IF NOT DTR$K_STL_PPUT THEN PERFORM 300-RECORD-NOT-STORED
                UNTIL DTR$K_STL_PPUT.
        GO TO 100-STORE-A-RECORD.

200-GET-A-RECORD.

**********************
* Clear the screen.  *
**********************
        MOVE 1 TO LINE_POS.
        MOVE 1 TO COL_POS.
        CALL "LIB$ERASE_PAGE" USING LINE_POS COL_POS.

**********************
* Format the screen. *
**********************

        MOVE "Enter a carriage return after each field value."
                TO TXT.
        MOVE 1 TO LINE_POS.
        MOVE 6 TO COL_POS.
        CALL "LIB$PUT_SCREEN" USING BY DESCRIPTOR TXT
                        BY REFERENCE LINE_POS COL_POS.

        MOVE SPACES TO TXT.
        MOVE "To stop storing records enter ALL DONE."
                TO TXT.
        MOVE 3 TO LINE_POS.
        MOVE 8 TO COL_POS.
        CALL "LIB$PUT_SCREEN" USING BY DESCRIPTOR TXT
                BY REFERENCE LINE_POS COL_POS.

        MOVE 6 TO LINE_POS.
        MOVE 25 TO COL_POS.
        CALL "LIB$PUT_SCREEN" USING BY DESCRIPTOR HDR_TXT
                BY REFERENCE LINE_POS COL_POS BOLD_VIDEO.
```

```
MOVE 9 TO LINE_POS.
MOVE 10 TO COL_POS.
CALL "LIB$PUT_SCREEN" USING BY DESCRIPTOR FNAME_TXT
                BY REFERENCE LINE_POS COL_POS REVERSE_VIDEO.
MOVE 9 TO LINE_POS.
MOVE 27 TO COL_POS.
MOVE "          " TO TXT.
CALL "LIB$PUT_SCREEN" USING BY DESCRIPTOR LNAME_TXT
                BY REFERENCE LINE_POS COL_POS REVERSE_VIDEO.

MOVE 40 TO COL_POS.
MOVE "(FIRST LAST)" TO TXT.
CALL "LIB$PUT_SCREEN" USING BY DESCRIPTOR TXT
                BY REFERENCE LINE_POS COL_POS.
MOVE 11 TO LINE_POS.
MOVE 8 TO COL_POS.
CALL "LIB$PUT_SCREEN" USING BY DESCRIPTOR STAT_TXT
                BY REFERENCE LINE_POS COL_POS REVERSE_VIDEO.

MOVE 40 TO COL_POS.
MOVE "(TRAINEE or EXPERIENCED)" TO TXT.
CALL "LIB$PUT_SCREEN" USING BY DESCRIPTOR TXT
                BY REFERENCE LINE_POS COL_POS.

MOVE 12 TO LINE_POS.
MOVE 12 TO COL_POS.
CALL "LIB$PUT_SCREEN" USING BY DESCRIPTOR ID_TXT
                BY REFERENCE LINE_POS COL_POS REVERSE_VIDEO.

MOVE 13 TO LINE_POS.
MOVE 4 TO COL_POS.
CALL "LIB$PUT_SCREEN" USING BY DESCRIPTOR DEPT_TXT
                BY REFERENCE LINE_POS COL_POS REVERSE_VIDEO.

MOVE 14 TO LINE_POS.
MOVE 1 TO COL_POS.
CALL "LIB$PUT_SCREEN" USING BY DESCRIPTOR SUP_TXT
                BY REFERENCE LINE_POS COL_POS REVERSE_VIDEO.

MOVE 15 TO LINE_POS.
MOVE 4 TO COL_POS.
CALL "LIB$PUT_SCREEN" USING BY DESCRIPTOR DATE_TXT
                BY REFERENCE LINE_POS COL_POS REVERSE_VIDEO.
MOVE 40 TO COL_POS.
MOVE "(EXAMPLE: 01-Jan-1982)" TO TXT.
CALL "LIB$PUT_SCREEN" USING BY DESCRIPTOR TXT
                BY REFERENCE LINE_POS COL_POS.

MOVE 16 TO LINE_POS.
MOVE 8 TO COL_POS.
CALL "LIB$PUT_SCREEN" USING BY DESCRIPTOR SAL_TXT
                BY REFERENCE LINE_POS COL_POS REVERSE_VIDEO.
```

(continued on next page)

```
*****************
* Accept input. *
*****************

         MOVE 9 TO LINE_POS.
         MOVE 16 TO COL_POS.
         CALL "LIB$PUT_SCREEN" USING BY DESCRIPTOR SPACE_TXT
                         BY REFERENCE LINE_POS COL_POS.
         ACCEPT FIRST_NAME.
         IF FIRST_NAME = "ALL DONE" THEN GO TO 999-EOJ.

         MOVE 27 TO COL_POS.
         CALL "LIB$PUT_SCREEN" USING BY DESCRIPTOR SPACE_TXT
                         BY REFERENCE LINE_POS COL_POS.
         ACCEPT LAST_NAME.

         MOVE 11 TO LINE_POS.
         MOVE 16 TO COL_POS.
         CALL "LIB$PUT_SCREEN" USING BY DESCRIPTOR SPACE_TXT
                         BY REFERENCE LINE_POS COL_POS.
         ACCEPT EMPLOYEE_STATUS.

         MOVE 12 TO LINE_POS.
         CALL "LIB$PUT_SCREEN" USING BY DESCRIPTOR SPACE_TXT
                         BY REFERENCE LINE_POS COL_POS.
         ACCEPT ID.

         MOVE 13 TO LINE_POS.
         CALL "LIB$PUT_SCREEN" USING BY DESCRIPTOR SPACE_TXT
                         BY REFERENCE LINE_POS COL_POS.
         ACCEPT DEPT.

         MOVE 14 TO LINE_POS.
         CALL "LIB$PUT_SCREEN" USING BY DESCRIPTOR SPACE_TXT
                         BY REFERENCE LINE_POS COL_POS.
         ACCEPT SUP_ID.

         MOVE 15 TO LINE_POS.
         CALL "LIB$PUT_SCREEN" USING BY DESCRIPTOR SPACE_TXT
                         BY REFERENCE LINE_POS COL_POS.
         ACCEPT START_DATE.

         MOVE 16 TO LINE_POS.
         CALL "LIB$PUT_SCREEN" USING BY DESCRIPTOR SPACE_TXT
                         BY REFERENCE LINE_POS COL_POS.
         ACCEPT SALARY.

300-RECORD-NOT-STORED.

         DISPLAY "Record was not stored.".
         DISPLAY "Press RETURN to continue.".
         ACCEPT CONT.
         CALL "DTR$COMMAND" USING DAB
                         BY DESCRIPTOR COMMAND_LINE.
```

```
999-EOJ.
***************************
* User entered ALL DONE.   *
***************************

        MOVE 1 TO LINE_POS.
        MOVE 1 TO COL_POS.
        CALL "LIB$ERASE_PAGE" USING LINE_POS COL_POS.

***************************
* Stop storing records.    *
***************************

        CALL "DTR$PORT_EOF" USING DAB.

        CALL "DTR$FINISH" USING DAB.
        DISPLAY "End of program.".
        STOP RUN.
```

4.2 A Sample Payroll Application

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

The program uses the data files TIMECARD.DAT and LABOR.DAT. These files are not included with the sample material in the DATATRIEVE kit. To use the sample payroll and update applications, define the two files and populate them with sample data. The domain and record definitions for HOURLY_LABOR are:

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

RECORD HOURLY_LABOR_REC USING
01  PERSON.
    05 ID                   PIC IS 9(5).
    05 EMPLOYEE_NAME        QUERY_NAME IS NAME.
        10 FIRST_NAME           PIC IS X(10)
                                QUERY_NAME IS F_NAME.
        10 LAST_NAME            PIC IS X(10)
                                QUERY_NAME IS L_NAME.
    05 DEPT                 PIC IS XXX.
    05 START_DATE           USAGE IS QUAD
                            DEFAULT VALUE IS "TODAY".
    05 HOURLY_RATE          PIC IS 99.99
                            EDIT_STRING IS $$$.99.
    05 SUP_ID               PIC IS 9(5)
                            MISSING VALUE IS 0.
;
```

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

```
PORT H_LABOR_PORT USING HOURLY_LABOR_REC;
```

PAYROLL works as follows:

1. The program reads data from TIMECARD.DAT.
2. The program uses H_LABOR_PORT to pass records from DATATRIEVE to a record buffer.
3. The program writes production data from TIMECARD.DAT out to the file FINISHED.DAT.
4. The program uses data from the DATATRIEVE record and TIMECARD.DAT to calculate the weekly employee salary.
5. Salary and other employee data are written to the file PAYROLL.LOG and to the screen.

Here is the program PAYROLL:

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

****************************************************
* The program reads data from a sequential file,   *
* TIMECARD.DAT. The program then uses information  *
* from a DATATRIEVE domain to create 2 log files:  *
* PAYROLL.LOG and FINISHED.DAT.                    *
****************************************************

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

INPUT-OUTPUT SECTION.

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

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

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

```
DATA DIVISION.
FILE SECTION.

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

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

FD  TIME-CARD-FILE
    RECORD IS VARYING IN SIZE FROM 19 TO 117 CHARACTERS.
01  TIME-REC.
    03  T-EMPLOYEE-NUMBER      PIC 9(5).
    03  T-JOB-COUNT            PIC 99.
    03  T-JOB-INFO  OCCURS 1 TO 10 TIMES
            DEPENDING ON T-JOB-COUNT.
        05  T-PRODUCT-NUMBER       PIC X(9).
        05  T-PRODUCT-HRS          PIC 99.

WORKING-STORAGE SECTION.
01  TIME-STATUS                PIC XX  VALUE SPACES.
01  FNSH-GDS-STATUS            PIC XX  VALUE SPACES.
01  PAYROLL-STATUS             PIC XX  VALUE SPACES.

01  SUB1                       PIC 999 COMP VALUE ZEROES.
01  TOTAL-HOURS                PIC 99.
01  OVERTIME-PAY               PIC 9999V99.
01  GROSS-PAY                  PIC 9999V99.
01  COUNTER                    PIC 99.

01  LINENO                     PIC 9 COMP.
01  COLNO                      PIC 9 COMP.
01  WS-TEXT                    PIC X(16) VALUE '         WORKING'.

*****************************************************
* Set up a record buffer by copying the CDD record  *
* definition.                                       *
*****************************************************
```

(continued on next page)

```
COPY "HOURLY_LABOR_REC" FROM DICTIONARY.

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

 COPY "DTR$LIBRARY:DAB.LIB".

01  DTR_OPTIONS PIC 9(9) COMP.
01  DTR_RETURN_STATUS PIC S9(9) COMP.

01  STACK-SIZE PIC 99 COMP VALUE 100.
01  COMMAND-LINE PIC X(80) VALUE
    "READY HOURLY_LABOR; ".

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

010-INITIALIZE-INTERFACE.

*********************
* Clear the screen. *
*********************

        MOVE 1 TO LINENO.
        MOVE 1 TO COLNO.
        CALL "LIB$ERASE_PAGE" USING LINENO COLNO.
        DISPLAY WS-TEXT.

********************************************
* Initialize the interface with DATATRIEVE *
********************************************

        CALL "DTR$INIT" USING DAB STACK-SIZE
                        BY DESCRIPTOR MSG_BUFF AUX_BUFF
                        GIVING DTR_RETURN_STATUS.

********************************************************
* Verify that DATATRIEVE was initialized successfully. *
********************************************************

        IF DTR_RETURN_STATUS IS FAILURE
        THEN
                DISPLAY "DATATRIEVE INITIALIZATION FAILED"
                GO TO 999-BAD-INIT.

***********************************************
* Use DTR$COMMAND to ready domains and ports. *
***********************************************

        CALL "DTR$COMMAND" USING DAB
                        BY DESCRIPTOR COMMAND-LINE.
```

```
***********************************************
* Use DTR$DTR to print out any print lines or *
* error messages and return on DTR$K_STL_CMD. *
***********************************************

        CALL "DTR$DTR" USING DAB DTR$M_OPT_CMD.
        MOVE "READY H_LABOR_PORT WRITE;" TO COMMAND-LINE.
        CALL "DTR$COMMAND" USING DAB
                        BY DESCRIPTOR COMMAND-LINE.
        CALL "DTR$DTR" USING DAB DTR$M_OPT_CMD.

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

030-GET-EMPLOYEE-RECORD.
**************************************************************
* Make sure that DATATRIEVE is at the command level stallpoint. *
**************************************************************

        IF NOT DTR$K_STL_CMD CALL "DTR$UNWIND" USING DAB.
        CALL "DTR$DTR" USING DAB DTR$M_OPT_CMD.

**********************************************************
* Pass a DATATRIEVE statement that will find all employees *
* with an employee number that matches T-EMPLOYEE-NUMBER    *
* in the file TIMECARD.DAT.                                *
**********************************************************

        MOVE "FOR HOURLY_LABOR WITH ID EQ !VAL " TO COMMAND-LINE.
        CALL "DTR$COMMAND" USING DAB
                        BY DESCRIPTOR COMMAND-LINE T-EMPLOYEE-NUMBER.

***********************************************************
* Command DATATRIEVE to store records from the HOURLY_LABOR  *
* domain into a port. DATATRIEVE stalls at DTR$K_STL_PGET    *
***********************************************************

        MOVE "STORE H_LABOR_PORT USING PERSON = PERSON;"
                        TO COMMAND-LINE.
        CALL "DTR$COMMAND" USING DAB
                        BY DESCRIPTOR COMMAND-LINE.

*************************************************
* Use DTR$DTR to print out messages and return on  *
* DTR$K_STL_CMD or DTR$K_STL_PGET.                 *
*************************************************
        ADD DTR$M_OPT_CMD
            DTR$M_OPT_PGET
                GIVING DTR_OPTIONS.
        CALL "DTR$DTR" USING DAB DTR_OPTIONS.
        IF NOT DTR$K_STL_PGET GO TO 100-NO-EMPLOYEE.
```

(continued on next page)

```
******************************************************
* Use the DTR$GET_PORT call to read an HOURLY_LABOR  *
* record into the program.                           *
******************************************************

        CALL "DTR$GET_PORT" USING DAB PERSON.
        CALL "DTR$DTR" USING DAB DTR_OPTIONS.


040-STORE-FINISHED-GOODS.
************************************************************
* Move the PRODUCT-NUMBER and the number of hours worked   *
* into FINISHED-REC. Write the data out to FINISHED.DAT    *
************************************************************

        MOVE T-PRODUCT-NUMBER (SUB1) TO F-PRODUCT-NUMBER.
        MOVE T-PRODUCT-HRS (SUB1) TO F-JOB-HRS.
        MULTIPLY T-PRODUCT-HRS (SUB1) BY HOURLY-RATE GIVING
           F-JOB-COST.
        WRITE FINISHED-REC.
        ADD  T-PRODUCT-HRS (SUB1) TO TOTAL-HOURS.
        SUBTRACT 1 FROM SUB1.
050-WRITE-PAYROLL-LOG.
************************************************************
* If number of hours is greater than 40, add overtime pay. *
* Move the data into PAY-REC and write PAYROLL.LOG.        *
***********************************************************

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

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

100-NO-EMPLOYEE.
*************************************************
* Alert operator if employee number is invalid. *
*************************************************
        MOVE DTR$M_OPT_CMD TO DTR_OPTIONS.
        CALL "DTR$DTR" USING DAB DTR_OPTIONS.
        DISPLAY "NO EMPLOYEE WITH THIS NUMBER, CHECK IT".
        DISPLAY T-EMPLOYEE-NUMBER.
        GO TO 020-READ-TIME-CARD-FILE.
```

```
999-EOJ.
*****************************************
* End interface, close files, and stop. *
*****************************************
        CALL "DTR$FINISH" USING DAB.
        CLOSE FINISHED-GOODS.
        CLOSE TIME-CARD-FILE.
        CLOSE PAYROLL-LOG-FILE.
        DISPLAY "   ".

999-BAD-INIT.
        DISPLAY "END OF PAYROLL UPDATE PROGRAM".
        STOP RUN.
```

4.3 A Sample Update Application

The program UPDATE uses the FINISHED.DAT file created by the PAYROLL program to update a DATATRIEVE domain named YEAR_TO_DATE_COST. The domain and record definitions for YEAR_TO_DATE_COST are:

```
DOMAIN YEAR_TO_DATE_COST USING YEAR_TO_DATE_COST_REC ON
YEARCOST.DAT;

RECORD YEAR_TO_DATE_COST_REC USING
01  YEAR_TO_DATE_COST.
    03  PRODUCT_NAME           PIC X(9).
    03  TOTAL_JOB_HRS          PIC 999V9
        EDIT_STRING IS ZZ9V9.
    03  TOTAL_JOB_COST         PIC 9(4)V99
        EDIT_STRING IS $ZZZ9V99.
;
```

This update application uses the data file YEARCOST.DAT. This file is not included with the sample material in the DATATRIEVE kit. To use the sample update application, define the file YEARCOST.DAT and populate it with sample data. The program displays the records it modifies and stores. It also displays a description of each product retrieved from the DATATRIEVE table PRODUCT_TABLE. The definition of PRODUCT_TABLE is:

```
TABLE PRODUCT_TABLE
        "VT125"  : "ADVANCED GRAPHICS TERMINAL",
        "VT100"  : "ADVANCED VIDEO OPTIONS TERMINAL",
        "VT52"   : "NO VIDEO OPTIONS TERMINAL"
        "LA36"   : "HARD COPY TERMINAL"
        ELSE " NOT FOUND IN TABLE"
END_TABLE
```

Here is the program UPDATE:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.          UPDATE01.

*****************************************************************
* The program reads the file FINISHED.DAT created by the PAYROLL *
* program. It uses the values read in to modify several fields   *
* in the DATATRIEVE domain YEAR_TO_DATE_COST.                    *
*****************************************************************

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

INPUT-OUTPUT SECTION.

********************************************************
* FINISHED.DAT contains weekly costs for each product.  *
********************************************************

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

DATA DIVISION.

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

WORKING-STORAGE SECTION.

COPY "DTR$LIBRARY:DAB.LIB".

* Return status for the file FINISHED.DAT.

01  FNSH-GDS-STATUS            PIC XX  VALUE SPACES.

* Return status for DATATRIEVE calls.

01  DTR-RETURN-STATUS          PIC S9(9) COMP.

* Return status for the paragraph 200-CHECK-FOR-FOUND.
```

```
01  FOUND-RECORD-STATUS  PIC S9(9) COMP.

01  CTR                  PIC 99 VALUE ZEROS.
01  WS_JOB_HRS           PIC 9(3).9.
01  WS_JOB_COST          PIC 9(4).9(2).
01  WS-PRODUCT-NAME.
    03  WS-Q1            PIC X VALUE """".
    03  WS-PN            PIC X(9).
    03  WS-Q2            PIC X VALUE """".
01  STACK-SIZE           PIC 99 COMP VALUE 100.
01  DTR_OPTIONS          PIC 9(9) COMP.
01  COMMAND-LINE         PIC X(80)
        VALUE "DECLARE PORT PT1 01 NUM PIC 99.;".
01  LINENO               PIC 9 USAGE IS COMP.
01  COLNO                PIC 9 USAGE IS COMP.
01  COMMENT-LINE.
    03  C-PRODUCT-NAME   PIC X(9).
    03  FILLER           PIC X.
    03  C-MESSAGE        PIC X(50).

PROCEDURE DIVISION.

000-OPEN-FILES.
    OPEN INPUT FINISHED-GOODS
*********************
* Clear the screen. *
*********************

    MOVE 1 TO LINENO.
    MOVE 1 TO COLNO.
    CALL "LIB$ERASE_PAGE" USING LINENO, COLNO.

010-INITIALIZE-INTERFACE.
***************************************************************
* Initialize the interface with DTR$INIT.                     *
***************************************************************

    CALL "DTR$INIT" USING DAB STACK-SIZE
                BY DESCRIPTOR MSG_BUFF AUX_BUFF
                BY REFERENCE DTR$K_SEMI_COLON_OPT
                GIVING DTR_RETURN_STATUS.

******************************************************
* Verify that DATATRIEVE was initialized successfully. *
******************************************************

    IF DTR_RETURN_STATUS IS FAILURE
        THEN
           DISPLAY "DATATRIEVE INITIALIZATION FAILED"
           GO TO 999-FINISHED.

    DISPLAY "  UPDATE Program working ".
```

(continued on next page)

```
****************************************************************
* Declare the options to be used in the DTR$DTR call.          *
****************************************************************

    ADD DTR$M_OPT_CMD DTR$M_OPT_PGET GIVING DTR_OPTIONS.
*************************************************************
* Declare a port to store the count of the collection found. *
* Verify the record was found.                               *
*************************************************************

    CALL "DTR$COMMAND" USING DAB
                       BY DESCRIPTOR COMMAND-LINE.
    CALL "DTR$DTR" USING DAB DTR$M_OPT_CMD.

***************************************
* Ready the YEAR_TO_DATE_COST domain. *
***************************************

    MOVE "READY YEAR_TO_DATE_COST WRITE;" TO COMMAND-LINE.
    CALL "DTR$COMMAND" USING DAB
                BY DESCRIPTOR COMMAND-LINE.
    CALL "DTR$DTR" USING DAB DTR$M_OPT_CMD.

*********************************************
* Read a record from the FINISHED.DAT file. *
*********************************************

050-READ-FINISHED-GOODS-FILE.
    READ FINISHED-GOODS AT END
            GO TO 950-REPORT.
    PERFORM 100-FIND-RECORD.
    PERFORM 200-CHECK-FOR-FOUND.

**********************************************
* If the record is not in the domain, add it; *
* otherwise, update the record.               *
**********************************************

    IF FOUND-RECORD-STATUS IS FAILURE THEN
        PERFORM 400-ADD-TO-DOMAIN
    ELSE
        PERFORM 300-UPDATE-RECORD.
    GO TO 050-READ-FINISHED-GOODS-FILE.

100-FIND-RECORD.

    MOVE SPACES TO COMMAND-LINE.
**********************************************************
* Put the quotation marks required by DATATRIEVE around   *
* the product name.                                       *
**********************************************************
```

```
    MOVE F-PRODUCT-NAME TO WS-PN.

*************************************************************
* Find the record whose product name matches the product name *
* read in from FINISHED.DAT.                                *
*************************************************************

    MOVE "FIND YEAR_TO_DATE_COST WITH PRODUCT_NAME = !CMD ;"
        TO COMMAND-LINE.
    CALL "DTR$COMMAND" USING DAB
        BY DESCRIPTOR COMMAND-LINE WS-PRODUCT-NAME.

********************************************************
* Skip the [# Records found] message and call DTR$DTR to *
* return control to the program at DTR$K_STL_CMD.      *
********************************************************

    CALL "DTR$CONTINUE" USING DAB.
    CALL "DTR$DTR" USING DAB DTR$M_OPT_CMD.

200-CHECK-FOR-FOUND.
    MOVE ZEROS TO CTR.

*******************************************************************
* Store the count of the number of records in the current collection *
* into a port.                                                    *
*******************************************************************

    MOVE "STORE PT1 USING NUM = COUNT;" TO COMMAND-LINE.
    CALL "DTR$COMMAND" USING DAB
                BY DESCRIPTOR COMMAND-LINE.
    CALL "DTR$DTR" USING DAB DTR_OPTIONS.

**************************************************************
* If the stallpoint is DTR$K_STL_PGET, retrieve the count; else *
* set FOUND-RECORD-STATUS to failure.                        *
**************************************************************

    IF DTR$K_STL_PGET THEN
        CALL "DTR$GET_PORT" USING DAB CTR
        CALL "DTR$DTR" USING DAB DTR$M_OPT_CMD
    ELSE
        SET FOUND-RECORD-STATUS TO FAILURE.

***************************************************************
* If one record was found, return success status; else return    *
* failure status.                                               *
***************************************************************

    IF CTR IS EQUAL TO 1 THEN
        SET FOUND-RECORD-STATUS TO SUCCESS
    ELSE
        SET FOUND-RECORD-STATUS TO FAILURE.
```

(continued on next page)

```
300-UPDATE-RECORD.

***************************
* Select the record found. *
***************************

    MOVE "SELECT;" TO COMMAND-LINE.
    CALL "DTR$COMMAND" USING DAB
                BY DESCRIPTOR COMMAND-LINE.
    CALL "DTR$DTR" USING DAB DTR$M_OPT_CMD.

*******************************************************************
* Instruct DATATRIEVE to modify the selected record by adding the  *
* values read in from FINISHED.DAT to the values currently stored. *
*******************************************************************

    MOVE F-JOB-COST TO WS-JOB-COST.
    MOVE F-JOB-HRS TO WS-JOB-HRS.
    MOVE "MODIFY USING BEGIN TOTAL_JOB_COST = TOTAL_JOB_COST + !VAL;"
                TO COMMAND-LINE.
    CALL "DTR$COMMAND" USING DAB
                BY DESCRIPTOR COMMAND-LINE WS-JOB-COST.
    CALL "DTR$DTR" USING DAB DTR$M_OPT_CMD.
    MOVE "TOTAL_JOB_HRS = TOTAL_JOB_HRS + !VAL; END;"
                TO COMMAND-LINE.
    CALL "DTR$COMMAND" USING DAB
                BY DESCRIPTOR COMMAND-LINE WS-JOB-HRS.
    CALL "DTR$DTR" USING DAB DTR$M_OPT_CMD.

******************************************************************
* This paragraph adds a record to the domain if it was not found. *
******************************************************************

400-ADD-TO-DOMAIN.

******************************************************************
* Display a message saying the product is not currently listed.  *
******************************************************************

    MOVE "NOT CURRENTLY LISTED IN THE DOMAIN" TO C-MESSAGE.
    MOVE F-PRODUCT-NAME TO C-PRODUCT-NAME.
    DISPLAY COMMENT-LINE.

***********************************
* Store the record in the domain. *
***********************************

    MOVE F-JOB-HRS TO WS-JOB-HRS.
    MOVE F-JOB-COST TO WS-JOB-COST.
    MOVE "STORE YEAR_TO_DATE_COST USING BEGIN"
        TO COMMAND-LINE.
    CALL "DTR$COMMAND" USING DAB
                BY DESCRIPTOR COMMAND-LINE.
    MOVE "PRODUCT_NAME = !CMD ;" TO COMMAND-LINE.
    CALL "DTR$COMMAND" USING DAB
                BY DESCRIPTOR COMMAND-LINE WS-PRODUCT-NAME.
    MOVE "TOTAL_JOB_HRS = !VAL ;" TO COMMAND-LINE.
    CALL "DTR$COMMAND" USING DAB
                BY DESCRIPTOR COMMAND-LINE WS-JOB-HRS.
    MOVE "TOTAL_JOB_COST = !VAL ; END;" TO COMMAND-LINE.
    CALL "DTR$COMMAND" USING DAB
                BY DESCRIPTOR COMMAND-LINE WS-JOB-COST.
    CALL "DTR$DTR" USING DAB DTR$M_OPT_CMD.

*********************************************************
* If the STORE command was successful, display a message. *
*********************************************************

    IF DAB$L_CONDITION IS EQUAL TO DTR$_SUCCESS THEN
        DISPLAY " *******  ADDED  ******* ".

950-REPORT.

*****************************************************************
* Use the DATATRIEVE PRINT statement to print a current list of   *
* the domain records, along with an average cost per hour, and a  *
* brief description of the product retrieved with the DATATRIEVE  *
* table PRODUCT_TABLE.                                            *
*****************************************************************

    MOVE "FOR YEAR_TO_DATE_COST PRINT PRODUCT_NAME, TOTAL_JOB_COST, -"
        TO COMMAND-LINE.
    CALL "DTR$COMMAND" USING DAB
                BY DESCRIPTOR COMMAND-LINE.
    CALL "DTR$DTR" USING DAB DTR$M_OPT_CMD.
    MOVE "TOTAL_JOB_HRS,TOTAL_JOB_COST/TOTAL_JOB_HRS -" TO COMMAND-LINE.
    CALL "DTR$COMMAND" USING DAB
                BY DESCRIPTOR COMMAND-LINE.
    CALL "DTR$DTR" USING DAB DTR$M_OPT_CMD.
    MOVE "(""AVERAGE COST PER HOUR"") USING $$$V99, SKIP 2, COL 1, -"
        TO COMMAND-LINE.
    CALL "DTR$COMMAND" USING DAB
                BY DESCRIPTOR COMMAND-LINE.
    CALL "DTR$DTR" USING DAB DTR$M_OPT_CMD.
    MOVE
    """DESCRIPTION:"", COL 14, (PRODUCT_NAME VIA PRODUCT_TABLE), SKIP 2"
        TO COMMAND-LINE.
    CALL "DTR$COMMAND" USING DAB
                BY DESCRIPTOR COMMAND-LINE.
    CALL "DTR$DTR" USING DAB DTR$M_OPT_CMD.

999-FINISHED.
    CLOSE FINISHED-GOODS.
    DISPLAY "End of program UPDATE".
    STOP RUN.
```

Sample BASIC Programs 5

This chapter contains sample BASIC programs that call DATATRIEVE. These programs show how you can call DATATRIEVE to perform information management tasks. Copies of the programs are in the DTR$LIBRARY directory.

To run these programs, you must compile them and then link them with the DATATRIEVE shareable image:

```
$ BASIC COLUMNS
$ LINK COLUMNS, DTR/OPT
$ RUN COLUMNS
```

See Chapter 2 for more information on using an options file to link programs that call DATATRIEVE.

5.1 Formatting a Report

The program COLUMNS creates a 2-column report of data in a DATATRIEVE domain. The program prompts for the name of the domain, a record selection expression, and the names of the fields that you want in the report. The program then displays the report on the screen and writes it to the file REPORT.LIS.

You may want to edit the program and change parameters such as the number and width of columns and the number of lines per page. Here is the program COLUMNS:

```
100     %INCLUDE "DTR$LIBRARY:DABS"
        ! Declare the buffers for the report file.

        COMMON (RPT_BUFFERS) STRING BIG_BUFF(2%,35%) = 35, &
                                    HEADERS (5%) = 35

        ! Declare the initialization call as a function.
        ! Declare the success status.

        EXTERNAL INTEGER FUNCTION DTR$INIT
        EXTERNAL LONG CONSTANT SS$_NORMAL

        ! Declare the options for the DTR$DTR call.

        DECLARE INTEGER DTR_OPTIONS
        DTR_OPTIONS = DTR$M_OPT_CMD + DTR$M_OPT_LINE

        DECLARE INTEGER RET_STATUS

        ! Open the report file.

        OPEN "REPORT.LIS" FOR OUTPUT AS FILE #1%, &
        SEQUENTIAL VARIABLE, RECORDSIZE 80%

        ! 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)

        ! Verify that DATATRIEVE was initialized successfully.

        IF RET_STATUS <> SS$_NORMAL THEN
             PRINT "DATATRIEVE initialization failed."
             GOTO 8100

        ! Clear the screen.
        ! Show the user the domains available.

525     CALL LIB$ERASE_PAGE BY REF (1%, 1%)
550     CALL DTR$COMMAND (DAB BY REF, "SHOW DOMAINS")

        ! Use DTR$DTR to display the results of SHOW DOMAINS.

        CALL DTR$DTR BY REF (DAB, DTR$M_OPT_CMD)

        ! Prompt the user for domain name and ready the domain.
```

```
600     INPUT "What is the domain"; DOM$
        CALL DTR$COMMAND (DAB BY REF, "READY !CMD", DOM$)

        ! Use DTR$DTR to handle messages and return at DTR$K_STL_CMD

        CALL DTR$DTR BY REF (DAB, DTR$M_OPT_CMD)

        ! Check for errors.

        IF DAB$L_CONDITION <> DTR$_SUCCESS THEN
                CALL LIB$ERASE_PAGE BY REF (1%, 1%)
                PRINT "Try again....."
                GOTO 550

        ! Show the available fields for the domain chosen.

700     CALL DTR$COMMAND (DAB BY REF, "SHOW FIELDS !CMD", DOM$)
        CALL DTR$DTR BY REF (DAB, DTR$M_OPT_CMD)

        ! Prompt for an RSE and field names.

        PRINT "Finish the record selection expression."
        LINPUT "FIND "; A$
        LINPUT "Enter a list of field names"; B$

        ! Instruct DATATRIEVE to print the chosen fields.
        ! Use DTR$DTR to print out the record stream.

        CALL DTR$COMMAND (DAB BY REF, "FOR !CMD PRINT !CMD",&
            A$, B$)
        CALL DTR$DTR BY REF (DAB, DTR_OPTIONS)

        ! Check for errors.

        IF DAB$W_STATE = DTR$K_STL_CMD THEN
                PRINT " TRY AGAIN "
                GOTO 700


800     CALL DTR$CONTINUE (DAB BY REF)   ! Skip the first blank line.

        ! Set the counter of header lines to 0.

        INC% = 0%

        ! Move the header lines into the header buffer.

        WHILE DAB$W_MSG_LEN <> 0%
                INC% = INC% + 1%
                HEADERS(INC%) = MSG_BUFF
                CALL DTR$CONTINUE (DAB BY REF)
        NEXT
        CALL DTR$CONTINUE (DAB BY REF)  ! Skip the last blank line.

        ! Fill the report buffer one side at a time.
```

(continued on next page)

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

        ! Move the contents of the message buffer into the
        ! report buffer. If there are no more records, clear out
        ! the rest of the buffer.

                IF DAB$W_STATE = DTR$K_STL_LINE THEN
                        BIG_BUFF(I%,J%) = MSG_BUFF
                        CALL DTR$CONTINUE (DAB BY REF)
                ELSE BIG_BUFF(I%,J%) = "    "
2200        NEXT J%
2300    NEXT I%

        ! Display the report file.
        ! Print one set of headers if there is only one column.

2600    CALL LIB$ERASE_PAGE BY REF (1%, 1%)
        FOR K% = 1% TO INC%
            IF BIG_BUFF(2%,1%) = "     "  THEN &
                 PRINT #1%, HEADERS(K%)
                 PRINT HEADERS(K%)
            ELSE
                 PRINT #1%, HEADERS(K%) + "     " + HEADERS(K%)
                 PRINT HEADERS(K%) + "     " + HEADERS(K%)
2800    NEXT K%

        ! Insert 1 blank line after headers.

2850    PRINT  1%, "   "
2875    PRINT "   "

        ! Write out the report file REPORT.LIS.
        ! Stop when there are no more records for the left side.

2900    FOR J% = 1% TO 35%
            IF BIG_BUFF(1%, J%) = "     "  THEN &
                GO TO 2950
            ELSE
                PRINT #1%, BIG_BUFF(1%,J%) + "     " + BIG_BUFF(2%,J%)
                PRINT BIG_BUFF(1%,J%) + "     " + BIG_BUFF(2%,J%)
2950    NEXT J%

        ! At the end of the page, print a form feed.


3200    IF DAB$W_STATE = DTR$K_STL_MSG  THEN &
                CALL DTR$DTR BY REF (DAB, DTR_OPTIONS)
3300    IF DAB$W_STATE = DTR$K_STL_LINE THEN &
                PRINT #1%, "<FF>"
                GOTO 2000

        ! Finish the session with DATATRIEVE.

8000    CALL DTR$FINISH (DAB BY REF)
8100    END
```

5.2 Calculating a Linear Regression Equation

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

The program first prompts the user for the names of a domain and two fields. The program next prompts for a DATATRIEVE FIND statement. The FIND statement determines which records are used in the regression. The program then determines the regression coefficients and prints them. Finally, the program enables the user to see how close the relationship is to being linear by displaying the actual and estimated field values.

Here is the program LINEAR:

```
 50     ON ERROR GO TO 8000

        ! Include the DATATRIEVE Access Block.

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

        ! Declare the initialization call as a function.
        ! Declare the normal status code.

        EXTERNAL INTEGER FUNCTION DTR$INIT
        EXTERNAL LONG CONSTANT SS$_NORMAL

        DECLARE STRING ANSWER
        DECLARE INTEGER RET_STATUS

        ! Format the record buffer and declare a table.

        MAP (VALUEAREA) REAL VALUE1, VALUE2
        MAP (VALUEAREA) STRING VALUES = 8
        DIMENSION AVERAGES(2)

        ! Declare the DTR$DTR options and assign the values.

        DECLARE INTEGER DTR_OPTIONS
        DTR_OPTIONS = DTR$M_OPT_CMD + DTR$M_OPT_PGET

        ! Initialize the session with DATATRIEVE.

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

        ! Verify that the initialization succeeded.

        IF RET_STATUS <> SS$_NORMAL THEN
             PRINT "DATATRIEVE initialization failed."
             GOTO 8300

        ! Declare the ports to be used in the program.
```

(continued on next page)

```
750     CALL DTR$COMMAND (DAB BY REF, &
                          "DECLARE PORT PT1 01 N PIC 9(5) COMP.;")

        CALL DTR$DTR BY REF (DAB, DTR$M_OPT_CMD)

        CALL DTR$COMMAND (DAB BY REF, &
                          "DECLARE PORT PT2 01 WHOLE.")
        CALL DTR$COMMAND (DAB BY REF, &
                          "02 PART-A REAL. 02 PART-B REAL.;")

        ! Use DTR$DTR to print any messages and return control to
        ! the program on DTR$K_STL_CMD.

        CALL DTR$DTR BY REF (DAB, DTR$M_OPT_CMD)

        ! Show the user the available domains.

1000    CALL DTR$COMMAND (DAB BY REF, "SHOW DOMAINS")
        CALL DTR$DTR BY REF (DAB, DTR$M_OPT_CMD )

        ! Prompt the user for the domain to be used and ready it.

        INPUT "What is the name of the domain"; DOMAIN$
        CALL DTR$COMMAND (DAB BY REF, "READY !CMD",DOMAIN$)
        CALL DTR$DTR BY REF (DAB, DTR$M_OPT_CMD)

        ! Check if the domain was readied.

        IF DAB$L_CONDITION <> DTR$_SUCCESS
           THEN GOTO 1000

        ! Prompt the user to enter a command to form a collection.
        ! Pass the command string to DATATRIEVE.

2000    PRINT "Enter a command to form a collection"
        INPUT COMMAND$
        CALL DTR$COMMAND (DAB BY REF, COMMAND$)
        CALL DTR$DTR BY REF (DAB, DTR$M_OPT_CMD)
        IF DAB$L_CONDITION <> DTR$_SUCCESS
           THEN GOTO 2000

        ! Store the count of the number of records found into
        ! a port then pass it to the program in the variable COUNTER%.

2750    CALL DTR$COMMAND (DAB BY REF, "STORE PT1 USING N = COUNT")
        CALL DTR$GET_PORT BY REF (DAB, COUNTER%)
        CALL DTR$DTR BY REF (DAB, DTR_OPTIONS)

        ! If there are no records, return to the section that prompts
        ! for a collection.

        IF COUNTER% = 0% THEN GOTO 2000
```

```
2800    REM                                                                 &
!*********************************************************************&
!       Formulas used to find the linear equation:                      &
!                                                                       &
!       LINEAR EQUATION :  Y = bX + a                                   &
!                                                                       &
!       Equation to arrive at value for b:                              &
!                       (note:  E = summation                           &
!                               n = number of data elements used)       &
!                                                                       &
!                 b = E(X*Y) - n(average(X) * average(Y))               &
!                     ----------------------------------                &
!                     E(X**2) - n(average(x)**2)                        &
!                                                                       &
!       Equation to arrive at value for a:                              &
!                                                                       &
!                 a = average(Y) - (b * average(X))                     &
!                                                                       &
!**********************************************************************

        ! Show the user the available fields.

3000    CALL DTR$COMMAND (DAB BY REF, "SHOW FIELDS FOR !CMD",DOMAIN$)
        CALL DTR$DTR BY REF (DAB, DTR$M_OPT_CMD)

        ! Prompt the user for the names of the independent
        ! and dependent fields.

        INPUT "What is the name of the independent field"; FIELD1$
        INPUT "What is the name of the dependent field"; FIELD2$

        ! Store the total added value of the fields chosen into a
        ! port.

        CALL DTR$COMMAND (DAB BY REF, "STORE PT2 USING BEGIN")
        CALL DTR$COMMAND (DAB BY REF, "PART-A = TOTAL !CMD;", &
                          FIELD1$)
        CALL DTR$COMMAND (DAB BY REF, "PART-B = TOTAL !CMD; END;", &
                          FIELD2$)
        CALL DTR$DTR BY REF (DAB, DTR_OPTIONS)

        ! Use DTR$GET_PORT to retrieve totals from DATATRIEVE and
        ! store them in the variable VALUES.

3500    CALL DTR$GET_PORT BY REF (DAB , VALUES)
        CALL DTR$DTR BY REF (DAB, DTR$M_OPT_CMD)
        IF DAB$L_CONDITION <> DTR$_SUCCESS
           THEN GOTO 3000

        ! Find the average value of each field (total / number
        ! of records).
```

(continued on next page)

```
3750    AVERAGES(1) = VALUE1 / COUNTER%
        AVERAGES(2) = VALUE2 / COUNTER%

        ! One at a time, store the value of the fields in each
        ! record into port PT2.

        CALL DTR$COMMAND (DAB BY REF, &
                          "FOR CURRENT STORE PT2 USING BEGIN")
        CALL DTR$COMMAND (DAB BY REF, &
                          "PART-A = !CMD; PART-B = !CMD; END;", &
                          FIELD1$, FIELD2$)

        ! The DATATRIEVE statement storing a port caused the
        ! stallpoint DTR$K_STL_PGET.
        !
        ! Use DTR$GET_PORT to retrieve the values stored in the port.

4000    CALL DTR$GET_PORT BY REF (DAB , VALUES)

        ! Calculate the summations needed for the linear regression.

        SUMXY = SUMXY + (VALUE1 * VALUE2)              ! E(XY).
        SUMX2 = SUMX2 + (VALUE1**2)                    ! E(X**2).

        ! Continue retrieving values until the whole collection has been
        ! stored.

        IF DAB$W_STATE = DTR$K_STL_PGET THEN GOTO 4000

4500    CALL DTR$DTR BY REF (DAB, DTR$M_OPT_CMD)

        ! Calculate the numerator and denominator of the linear
        ! regression equation.

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

        ! Calculate the values of the slope (b) and y-intercept (a).

        b = TOP/BOTTOM
        a = AVERAGES(2) - (b * AVERAGES(1))

        ! Print out the linear equation.

        PRINT "Best estimate for linear relation is..."
        PRINT FIELD2$; " = "; a; " + "; b ; " * "; FIELD1$

        ! Ask if the user wishes to see the calculated values and the
        ! actual values.

        INPUT "Enter Y if you want to see relationship"; ANSWER
        IF ANSWER = "Y" THEN GOSUB 6000

        ! Prompt for what the user wishes to do next.
```

```
5000    PRINT "Enter CTRL/Z to exit program."
        PRINT "Enter 1 to start over with new domain."
        PRINT "Enter 2 to start over with new collection."
        PRINT "Enter 3 to use same collection, different fields."

        INPUT D%

        ! If the user did not enter 1, 2, or 3, prompt again.

5250    IF (D% > 3%) OR (D% < 1%)
           THEN PRINT "Invalid entry, try again."
                GOTO 5000

        ! Clear all variables.

5500    VALUE1 = 0.0
        VALUE2 = 0.0
        AVERAGES(1) = 0.0
        AVERAGES(2) = 0.0
        SUMXY = 0.0
        SUMX2 = 0.0
        a = 0.0
        b = 0.0

        ON D% GOTO 1000, 2000, 3000


        ! Print the values of the two fields for the current
        ! collection.

6000    CALL DTR$COMMAND (DAB BY REF,"FOR CURRENT PRINT !CMD, !CMD,", &
                          FIELD1$, FIELD2$)

        ! Command DATATRIEVE to print the values calculated by the
        ! equation.

        CALL DTR$COMMAND (DAB BY REF, &
                          '!VAL + !VAL * !CMD ("ESTIMATE")', &
                          A BY DESC, B BY DESC, FIELD1$)

        CALL DTR$COMMAND (DAB BY REF, " ON TT: ")
        CALL DTR$DTR BY REF (DAB, DTR$M_OPT_CMD)
        RETURN

        ! If the user enters CTRL/Z, end the interface and exit program.

8000    IF ERR = 11% THEN RESUME 8200

8100                                ON ERROR GOTO 0

8200    CALL DTR$DTR BY REF (DAB, DTR$M_OPT_CMD)
        CALL DTR$FINISH (DAB BY REF)

8300    END
```

Reference Section

Replace this page with heavy page entitled Reference Section located at the end of this package.

DATATRIEVE Call Reference 6

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:

  - 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. See the programs in Chapters 3, 4, and 5 for more examples.

6.1 DTR$COMMAND – Passing Commands and Statements to DATATRIEVE

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. If you use the inclusion file provided in your installation kit, the name is DAB.

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 substitution directives. The two substitution directives are:

!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

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 or DTR$DTR call before passing your next DTR$COMMAND call.
- For each !CMD and !VAL substitution directive in the command string, you must include an argument for DATATRIEVE to substitute.
- Use the !CMD substitution directive only to insert strings in command strings and the !VAL substitution directive only to insert numeric values in command strings.
- When storing a procedure in the CDD, 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")
```

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 three 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.

When writing programs that create DATATRIEVE procedures, you must be very careful to design the callable program so that the use of substitution strings will not cause premature termination of the DATATRIEVE command or statement.

- 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. See the PARSE subroutine in Chapter 3 for an example that uses the DTR$L_MORE_COMMANDS option to pass a long command string.

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

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:

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

Pass a command to DATATRIEVE from a BASIC program:

```
800     CALL 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.
        CALL "DTR$COMMAND" USING DAB
                 BY DESCRIPTOR COMMAND_LINE.
```

Enter commands when you run your program:

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

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%

        CALL 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;';
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)
```

(continued on next page)

```
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.

            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.

            CALL 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.

            CALL 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     CALL DTR$COMMAND (DAB, 'READY YACHTS')

C Print out any messages and return on command stallpoint.

        CALL DTR$DTR (DAB, DTR$M_OPT_CMD)

        DO WHILE (.TRUE.)

C Show the fields available for the domain YACHTS.

            CALL DTR$COMMAND (DAB, 'SHOW FIELDS YACHTS')
            CALL DTR$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.

            CALL DTR$COMMAND (DAB, 'PRINT YACHTS WITH !CMD GT !VAL',
        1   FIELD,
        2   VALUE)

C Use DTR$DTR to print out the record stream.

            CALL DTR$DTR (DAB, DTR_OPTIONS)
        END DO

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

6.2 DTR$CONTINUE – Returning Control to DATATRIEVE

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. If you use the inclusion file provided in your installation kit, the name is DAB.

Usage Notes

- You can use the DTR$CONTINUE call when DATATRIEVE is at the continue, message, or print line stallpoint (DTRK_STL_CONT, DTRK_STL_MSG, or DTR$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 recover 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.

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 DTRK_STL_CONT, DTRK_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;
   BEGIN
   WHILE (DAB.DAB$W_STATE = DTR$K_STL_MSG) OR
         (DAB.DAB$W_STATE = DTR$K_STL_LINE) DO
      BEGIN
      WRITELN (MSG_BUFF);
      DTR$CONTINUE (DAB);
      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'

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:

        1 ((DAB$L_CONDITION .EQ. %LOC(DTR$_RECFOUND)) .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.

                CALL 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.

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

6.3 DTR$CREATE_UDK - Defining a Keyword

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. If you use the inclusion file provided in your installation kit, the name is DAB.

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 6-1 lists the user-defined keyword contexts and their values.

Table 6-1: 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$DTR to get access to interactive DATATRIEVE. If you set the DTR$DTR 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. The program CORRELATE in Chapter 3 shows 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-defined keywords.

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.
```

(continued on next page)

```
        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.

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    CALL DTR$CREATE_UDK (DAB BY REF, 'CLEAR_SCREEN', 1% BY REF, &
                             DTR$K_UDK_COMMAND BY REF)

        CALL DTR$CREATE_UDK (DAB BY REF, 'DIRECTORY', 2% BY REF, &
                             DTR$K_UDK_COMMAND BY REF)

        CALL DTR$CREATE_UDK (DAB BY REF, 'MAIL', 3% BY REF, &
                             DTR$K_UDK_COMMAND BY REF)

        CALL DTR$CREATE_UDK (DAB BY REF, 'SPAWN', 4% BY REF, &
                             DTR$K_UDK_COMMAND BY REF)

        CALL 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$W_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.

        CALL DTR$END_UDK BY REF (DAB)
        RETURN

        ! UDK 2 - User entered DIRECTORY.

3200    CALL LIB$SPAWN ("DIRECTORY")
        CALL DTR$END_UDK BY REF (DAB)
        RETURN

        ! UDK 3 - User entered MAIL.

3300    CALL LIB$SPAWN ("MAIL")
        CALL DTR$END_UDK BY REF (DAB)
        RETURN

        ! UDK 4 - User entered SPAWN.

3400    CALL LIB$SPAWN ()
        CALL DTR$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"

        CALL DTR$END_UDK BY REF (DAB)
        RETURN

6000    CALL DTR$FINISH BY REF (DAB)

7000    END
```

This program shows you how to add commands to DATATRIEVE by using Run-Time Library routines. For an example of how to add statements, see the program CORRELATE in Chapter 3.

Note that the DTR$M_OPT_FOREIGN option to the DTR$DTR 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
```

6.4 DTR$DTR - Using the DATATRIEVE Terminal Server

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.

Format

```
DTR$DTR  (dab,  options-code)
```

Arguments

dab

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

Is the DATATRIEVE Access Block. If you use the inclusion file provided in your installation kit, the name is DAB.

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$DTR options and list their values. Table 6-2 describes the DTR$DTR options that determine when the DATATRIEVE terminal server returns control to your program. Table 6-3 describes the DTR$DTR options that enable various DATATRIEVE terminal server functions.

Table 6-2: 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
DTR$M_OPT_PRMPT	2	Stallpoint is DTR$K_STL_PRMPT
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$M_OPT_DTR_UDK	256	User enters a DATATRIEVE-defined keyword
DTR$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 6-3: 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
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

- 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.

  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.

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

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

```
IF DTR$K_STL_LINE THEN
        CALL "DTR$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_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    CALL DTR$FINISH BY REF (DAB)

2000    END
```

You can use this program as the framework for customizing interactive DATATRIEVE. For more examples of how to use DTR$DTR, see Chapters 3, 4, and 5.

6.5 DTR$END_UDK - Terminating a User-Defined Keyword

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. If you use the inclusion file provided in your installation kit, the name is DAB.

Usage Notes

- To use this call you must be at the command, user-defined keyword, or end user-defined keyword stallpoint (DTRK_STL_CMD, DTRK_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. The program CORRELATE in Chapter 3 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 DTRK_STL_CMD, DTRK_STL_UDK, or DTR$K_STL_END_UDK stallpoint.

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.

        CALL 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.

                CALL LIB$SPAWN()
        END IF
```

(continued on next page)

```
C Terminate processing of the UDK.

        CALL DTR$END_UDK (DAB)

C Recall the terminal server.

        GO TO 100

999     CALL DTR$FINISH (DAB)

        END
```

6.6 DTR$FINISH – Ending a Program's Interaction with DATATRIEVE

Ends a program's interaction with DATATRIEVE.

Format

```
DTR$FINISH  (dab)
```

Argument

dab

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

Is the DATATRIEVE Access Block. If you use the inclusion file provided in your installation kit, the name is DAB.

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.

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.

        CALL DTR$FINISH (DAB)
```

6.7 DTR$GET_PORT - Passing Records to Your Program

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. If you use the inclusion file provided in your installation kit, the name is DAB.

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)
```

- If you write your programs in a high-level language that supports the VAX Common Data 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 CDD and use it as a record buffer.

  To include a CDD record in a COBOL program, use the COPY statement, as in the following example:

  ```
  COPY "CDD$TOP.COBOL.YACHT_COBOL" FROM DICTIONARY
  ```

  To include a CDD record in a BASIC program, use the %INCLUDE directive:

  ```
  %INCLUDE %FROM %CDD "CDD$TOP.DTR$LIB.DEMO.YACHT"
  ```

  To include a CDD record in a FORTRAN program, use the DICTIONARY statement:

  ```
  DICTIONARY 'CDD$TOP.DTR$LIB.DEMO.YACHT'
  ```

  In each case, 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 CDD. 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 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 CDD. You can issue the DECLARE PORT statement only from your program. You must define a record in the statement.

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

  ```
  CALL 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).
- You should be careful to use record buffers that are the same size as the ports. If the sizes do not match, then 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$_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$K_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 CDD 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).

PROCEDURE DIVISION.

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.                                   *
****************************************************************

        CALL "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.
        CALL "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.                                                  *
****************************************************************
```

(continued on next page)

```
        CALL "DTR$DTR"  USING DAB DTR$M_OPT_CMD.

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

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

        CALL "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.           *
*****************************************************************

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

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

        IF DTR$K_STL_MSG
        CALL "DTR$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.*
*****************************************************************

        CALL "DTR$GET_PORT" USING DAB BOAT.
        DISPLAY MANUFACTURER "   " MODEL "   " RIG "   "
        LENGTH_OVER_ALL "   " BEAM "     " DISPLACEMENT "    " PRICE.
```

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

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

6.8 DTR$GET _ STRING - Parsing the Arguments to a UDK

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. If you use the inclusion file provided in your installation kit, the name is DAB.

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 6-4 lists all the token types you can specify.

Table 6-4: 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$K_TOK_PICTURE	3	Retrieves a picture string
DTR$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

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, then 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.

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, then 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$M_OPT_UDK option to DTR$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
```

For a complete example of how to parse a string entered after a user-defined keyword, see the subroutine PARSE in Chapter 3.

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.

Example

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
```

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    CALL 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$M_OPT_UDK ! Return to program on UDK &
                         + DTR$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

3000    OPEN "TEMP.COM" FOR OUTPUT AS FILE #2%

        ! Use DTR$GET_STRING to get the procedure name.

        CALL 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.
```

(continued on next page)

```
4000    PRINT #2%, "$ RUN SYS$SYSTEM:DTR32"
        PRINT #2%, "EXECUTE ";DTR_PROCEDURE
        PRINT #2%, "$ EXIT"
        CLOSE 2%

        ! Submit the command file.

        CALL LIB$SPAWN ("SUBMIT/QUEUE=SYS$BATCH/NOPRINTER "+  &
                        "TEMP/NOTIFY/DELETE/LOG=[]")

        ! End the UDK.

        CALL DTR$END_UDK BY REF (DAB)

        ! Recall the terminal server.
        GO TO 2500

6000    CALL DTR$FINISH BY REF ( DAB )
        END
```

6.9 DTR$INFO - Getting Information About DATATRIEVE Objects

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:

- 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. If you use the inclusion file provided in your installation kit, the name is DAB.

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

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 6-5 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 an object-id or a number that provides your program with information. (See Table 6-5.)

output-string

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

Is a string DATATRIEVE uses to return the information you request. Table 6-5 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 6-5 lists:

- 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

Table 6-5: Info-Code Options

Info-code	Number of arguments required	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
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

(continued on next page)

Table 6-5: Info-Code Options (Cont.)

Info-code	Number of arguments required	DTR$INFO returns:
Information about domains		
Pass in an object-id obtained with DTR$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
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$K_INF_DOM_SSC	4	Object-id of subschema of DBMS domain
DTR$K_INF_DOM_REC_LEN	4	Length of record associated with current domain

(continued on next page)

Table 6-5: Info-Code Options (Cont.)

Info-code	Number of arguments required	DTR$INFO returns:
Information about fields		
Pass in the object-id of a top-level field obtained with DTR$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$K_INF_FLD_EDIT	5	Edit string for field in output-string
DTR$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)

(continued on next page)

Table 6-5: Info-Code Options (Cont.)

Info-code	Number of arguments required	DTR$INFO returns:
DTR$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$K_INF_FLD_MISSING	5	Descriptor of the MISSING VALUE
DTR$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

(continued on next page)

Table 6-5: Info-Code Options (Cont.)

Info-code	Number of arguments required	DTR$INFO returns:
DTR$K_INF_COL_NAME	5	Name of collection in output-string
DTR$K_INF_COL_NEXT_COL	4	Object-id of next collection in collection list
Information about forms		
Pass in the object-id of a form obtained with DTR$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 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

(continued on next page)

Table 6-5: Info-Code Options (Cont.)

Info-code	Number of arguments required	DTR$INFO returns:
DTR$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		
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)

Table 6-5: Info-Code Options (Cont.)

Info-code	Number of arguments required	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, and PL/I 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 (DTRK_STL_CMD, DTRK_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> 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 DTRK_STL_CMD, DTRK_STL_UDK, or DTR$K_STL_END_UDK.

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

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.

        CALL 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
C object-id of the first collection in the collection list.

        CALL 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.

        CALL 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.

        CALL 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

        CALL 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
```

Here is the program that 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.

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

C Create the UDK PLOT_ARGUMENTS.

        CALL DTR$CREATE_UDK (DAB, 'PLOT_ARGUMENTS', 1, DTR$K_UDK_COMMAND)

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

        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 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$W_UDK_INDEX .EQ. 1) .AND.
        1  (DAB$W_STATE .EQ. DTR$K_STL_UDK))

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

            CALL DTR$GET_STRING (DAB, DTR$K_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.

           CALL DTR$LOOKUP (DAB, DTR$K_INF_TYPE_PLOT, PLOT_ID,
        1                    PLOT_NAME)

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

             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.

           CALL 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        CALL DTR$END_UDK (DAB)
        END DO

        GO TO 10

999     CALL 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
```

(continued on next page)

```
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
```

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

        CALL 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.

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

        ! Get the query name of the field.

350     CALL 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.

        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 CALL 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%
```

(continued on next page)

```
        ! Get the object-id of the child field.

              CALL 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.

              CALL 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                                     &
!*********************************************************************

        ! 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    CALL DTR$CREATE_UDK (DAB BY REF, 'RECORD_LENGTH', 1%, &
                             DTR$K_UDK_SHOW)

        ! Use DTR$LOOKUP to get the global variable identifier.

        CALL 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.

          CALL 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.

            CALL 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.

            CALL DTR$INFO (DAB BY REF, DOM_ID%, DTR$K_INF_DOM_FLD, &
                           FLD_ID% , ,)

        ! Find the number of elementary fields the first field has.

            CALL DTR$INFO (DAB BY REF, FLD_ID%, DTR$K_INF_FLD_CNT, &
                           FIELD_COUNT%,,)
```

(continued on next page)

```
        ! 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.

            CALL 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        CALL DTR$INFO (DAB BY REF, DOM_ID%, DTR$K_INF_DOM_NEXT_DOM, &
                           DOM_ID% , ,)

          NEXT

          CALL 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    CALL DTR$FINISH BY REF ( DAB )
8000    END
```

6.10 DTR$INIT - Initializing the DATATRIEVE Call Interface

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. If you use the inclusion file provided in your installation kit, the name is DAB.

size

Data type: word 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

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:

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 will prompt 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 accesses domains which use a form or passes a DISPLAY_FORM statement, DATATRIEVE displays the form on your terminal screen. Setting this option is equivalent to using the SET FORM command.

DTR$K_VERIFY = 256

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. For an example of how to use this option, see the program CORRELATE in Chapter 3.

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.

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.

- 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.

- You can change options after initialization by changing the value of DAB$L__OPTIONS.

- DTR$INIT does not invoke the DATATRIEVE command startup file. See the call DTR$DTR for the option that invokes the startup file.

- If your program opens the DATATRIEVE Call Interface with a call to DTR$INIT and then closes the interface with a call to DTR$FINISH, the program must reinitialize the DAB before it can be used in another call to DTR$INIT. That is, the values of all the fields must be set to zero.
- Your program can start up to five DATATRIEVE sessions at one time. Each session requires a separate Access Block and a call to DTR$INIT to start the session. If you want to initialize DATATRIEVE more than five times, you must call DTR$FINISH to close one of the existing sessions, reinitialize the DAB, and then call DTR$INIT again to start the new session.

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    DTR$INIT
        INTEGER      RET_STATUS
        EXTERNAL     SS$_NORMAL
```

(continued on next page)

```
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$INIT (DAB, 100, MSG_BUFF, AUX_BUFF,
       1   DTR$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
```

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
```

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'

BEGIN

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

DTR$INIT (DAB, 100, MSG_BUFF, AUX_BUFF, INIT_OPTIONS);
```

6.11 DTR$LOOKUP - Looking Up DATATRIEVE Objects

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

- Find out if a CDD 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. If you use the inclusion file provided in your installation kit, the name is DAB.

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:

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

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, and PASCAL in the DTR$LIBRARY directory.
- You can use this call when DATATRIEVE is at the command, user-defined keyword, or end user-defined keyword stallpoint (DTRK_STL_CMD, DTRK_STL_UDK, or DTR$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 DTRK_STL_CMD, DTRK_STL_UDK, or DTR$K_STL_END_UDK.

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

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 CDD 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.

6.12 DTR$PORT_EOF - Ending the Transfer of Records to DATATRIEVE

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. If you use the inclusion file provided in your installation kit, the name is DAB.

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.

Example

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

```
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
```

(continued on next page)

```
C Ready domain.

        CALL DTR$COMMAND (DAB, 'READY YACHTS WRITE')

C Display messages.

        CALL DTR$DTR (DAB, DTR$M_OPT_CMD)

C Set up a port to pass records to DATATRIEVE.

        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).;')

C Display messages.

        CALL 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.

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

C Use DTR$PUT_PORT to pass the complete record.

        CALL DTR$PUT_PORT (DAB, %REF(YACHT))

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

        IF (DAB$W_STATE .EQ. DTR$K_STL_MSG)
        1 CALL 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     CALL DTR$PORT_EOF (DAB)
        CALL DTR$DTR (DAB, DTR$M_OPT_CMD)

C End the Interface.

        CALL DTR$FINISH (DAB)

        END
```

6.13 DTR$PRINT_DAB - Debugging Your Program

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. If you use the inclusion file provided in your installation kit, the name is DAB.

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, the condition, and the 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:

```
! 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:

```
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$DTR with the DTR$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$DTR between the first and second calls, DATATRIEVE still does not execute the PRINT statement. If you use the call DTR$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.

6.14 DTR$PUT_OUTPUT - Writing Output to a File

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. If you use the inclusion file provided in your installation kit, the name is DAB.

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.

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:

```
                .
                .
                .
        CALL 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.

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

        ! End the UDK.

        CALL DTR$END_UDK BY REF (DAB)

        GOTO 2500
6000    END
```

6.15 DTR$PUT_PORT - Passing Records to DATATRIEVE

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. If you use the inclusion file provided in your installation kit, the name is DAB.

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 CDD (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 CDD and use it as a record buffer.

  To include a CDD record in a COBOL program, use the COPY statement, as in the following example:

  ```
  COPY "CDD$TOP.COBOL.YACHT_COBOL" FROM DICTIONARY
  ```

  To include a CDD record in a BASIC program, use the %INCLUDE directive:

  ```
  %INCLUDE %FROM %CDD "CDD$TOP.DTR$LIB.DEMO.YACHT"
  ```

To include a CDD record in a FORTRAN program, use the DICTIONARY statement:

```
DICTIONARY 'CDD$TOP.DTR$LIB.DEMO.YACHT'
```

In each case, 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.

- The DEFINE PORT command inserts your port definition into the CDD. 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 CDD. 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$PORT_EOF.
- After each DTR$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.

6.16 DTR$PUT_VALUE - Transfer of Values

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. If you use the inclusion file provided in your installation kit, the name is DAB.

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$K_STL_PRMPT) and returns control to your program. Your program must provide a value for the prompting expression before DATATRIEVE can continue. To pass a value back to DATATRIEVE, use the call DTR$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"
  ```

  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_PRMPT

C Display the DATATRIEVE value prompt and read a value.

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

                CALL 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.

                CALL DTR$DTR (DAB, DTR_OPTIONS)
        END DO
```

6.17 DTR$UNWIND – Aborting Commands

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. If you use the inclusion file provided in your installation kit, the name is DAB.

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.

Example

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

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

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

        CALL 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.

        CALL DTR$DTR (DAB, DTR$M_OPT_CMD)

100     CALL 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_PRMPT   ! Return on DTR$K_STL_PRMPT

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

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.

                CALL DTR$PUT_VALUE (DAB, VALUE)
```

(continued on next page)

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

                IF(DAB$W_STATE .EQ. DTR$K_STL_MSG)  THEN
                        CALL 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
            CALL DTR$UNWIND (DAB)

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

            CALL DTR$PUT_VALUE (DAB, VALUE)

C Use DTR$DTR to display messages.

             CALL 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

        CALL DTR$FINISH (DAB)
        END
```

Adding Functions to DATATRIEVE 7

A DATATRIEVE **function** is a word 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 create and install a new shareable image. This requires the VMS privileges SYSPRV and CMKRNL.

7.1 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

        10
```

You can nest functions, as the following example shows:

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

              FN$HEX

        32
```

These examples 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>
```

7.2 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 sample 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.

7.3 How to Add Functions to DATATRIEVE

To add functions to DATATRIEVE, you must use several files in DTR$LIBRARY. These files are:

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

Note that these files may have a suffix appended to the file name. DTRBLD 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 DTRFUN.OLB.
3. Add the function definitions to DTRFND.MAR.
4. Assemble DTRFND.MAR, creating DTRFND.OBJ.
5. Replace DTRFND.OBJ in DTRFUN.OLB.
6. Relink the DATATRIEVE shareable image.

For additional information on creating and adding 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.

7.3.1 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 the section entitled Add the Function Definitions to DTRFND.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 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 VAX Linker makes FORTRAN and COBOL code position-independent. Therefore, you can write procedures in FORTRAN and COBOL and link them with DTRSHR.

Here is an example of a procedure 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:

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

The VAX Linker searches the module library DTR$LIBRARY:DTRFUN.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
```

7.3.2 Insert the Procedure Object Files into the Object Library

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

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

7.3.3 Add the Function Definitions to DTRFND.MAR

The file DTRFND.MAR contains the definitions of all the sample functions that are included in the DATATRIEVE installation kit. These definitions serve as examples of how to define functions.

DTRFND.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 DTRFND.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.

7.3.3.1 The Format of a Function Definition

7.3.3.1 The Format of a Function Definition — The format of a function definition is:

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

                          ⎧ FUN$K_STATUS                       ⎫
$DTR$FUN_OUT_ARG  TYPE =  ⎨ FUN$K_INPUT                        ⎬
                          ⎩ FUN$K_VALUE   , DTYPE = data-type  ⎭

⎡                         ⎛ FUN$K_NULL                                                          ⎞ ⎤
⎢                         ⎜ FUN$K_TEXT    , OUT_PUT = TRUE     [, ALL_LEN = m]                  ⎟ ⎥
⎢ $DTR$FUN_IN_ARG  TYPE = ⎨ 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
```

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 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

This section of a function definition 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.

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 7-1 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
```

This section of a function definition 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.

There must be one $DTR$FUN_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.

The argument-passing types you can specify with the TYPE parameter are:

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 ALL_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.

The clauses you use with the type parameter are:

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

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 7-1 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 OUT_PUT = TRUE clause 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.

Table 7-1 lists the data types commonly used in functions. For a complete list of the VAX data types refer to Appendix C.

Table 7-1: Common VAX Data Types

Symbol	Description
DSC$K_DTYPE_LU	Longword logical. A 32-bit unsigned quantity.
DSC$K_DTYPE_QU	Quadword logical. 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 7-2 shows which clauses are required or optional for each argument passing type.

Table 7-2: 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

7.3.3.2 Sample Function Definitions — The following function definition allows you to round off numbers within DATATRIEVE. The actual procedure used is MTH$JNINT, from the VAX Run-Time Library. Like most of the Run-Time Library mathematics procedures, MTH$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:

```
; 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
```

The $DTR$FUN_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

                               LENGTH
                                OVER
MANUFACTURER   MODEL     RIG    ALL    WEIGHT BEAM  PRICE

 ALBIN       79          SLOOP  26      4,200  10  $17,900
 ALBIN       BALLAD      SLOOP  30      7,276  10  $27,500
 ALBIN       VEGA        SLOOP  27      5,070  08  $18,600
 ELAPSED: 00:00:00.60  CPU: 0:00:00.20  BUFIO: 8  DIRIO: 3  FAULTS: 31
```

(continued on next page)

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

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

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 the section entitled 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
```

7.3.4 Assemble and Debug DTRFND

After you place function definitions in DTRFND.MAR, reassemble it with the command:

```
$ MACRO/LIST DTR$LIBRARY:DTRFND.MAR
```

If you have made any mistakes in your function definitions, you should get error messages when you assemble DTRFND.MAR. The following are explanations of these error messages:

Error 1: Function definition not in progress.

Occurs if you omit the $DTR$FUN_DEF section from your function definition.

Error 2: Duplicate definition within a single function.

Occurs if:

- You specify $DTR$FUN_DEF without a corresponding $DTR$FUN_END_DEF
- You have two $DTR$OUT_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 $DTR$FUN_DEF section.

Error 5: More input arguments than specified in function definition.

Occurs if you include more input arguments than you specify in the $DTR$FUN_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 $DTR$FUN_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 $DTR$FUN_OUT_ARG section.

Error 9: Output argument not found in input arguments.

Occurs if you do not specify the FUN$K_VALUE passing type for $DTR$FUN_OUT_ARG, and you also do not have any $DTR$FUN_IN_ARG sections with the OUT_PUT = TRUE clause.

If you get any of these messages when you assemble DTRFND.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.)

7.3.5 Replace DTRFND.MAR in the Object Library

After you successfully assemble DTRFND.MAR, use the LIBRARY command to put the new object file, DTRFND.OBJ, in the library DTRFUN.OLB:

```
$ LIBRARY/REPLACE DTR$LIBRARY:DTRFUN DTRFND
```

7.3.6 Relink the DATATRIEVE Shareable Image

After you replace DTRFND.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 DTRBLD.COM to relink the DATATRIEVE shareable image. Before invoking DTRBLD.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 DTRBLD.COM to include that RTL library. For example, to use the BASIC procedure POWER described in the section entitled Write and Compile Your Procedures and defined in the section entitled Sample Function Definitions, you must add these two lines to the DTRBLD.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
```

To relink VAX DATATRIEVE, execute the command file DTRBLD:

```
$ @DTR$LIBRARY:DTRBLD
```

Note that the DTRBLD.COM file may have a suffix appended to the file name. DTRBLD might be DTRBLDFMS, for example.

Customizing DATATRIEVE Help Text 8

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 online 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 online documentation needs.

8.1 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.

8.1.1 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.

8.1.2 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 DEFINE command, use the following LIBRARY command:

```
$ LIBRARY/HELP/EXTRACT=DEFINE/OUTPUT=DEFINE DTRHELP.HLB
```

The /EXTRACT qualifier causes the VAX Librarian to extract the DEFINE module from DTRHELP.HLB. The /OUTPUT qualifier places the HELP DEFINE text in the file DEFINE.HLP in your default VMS directory. Note that DEFINE.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.

8.1.3 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.

8.1.3.1 The Structure of Help Text Files — Help files have a hierarchical structure, much like DATATRIEVE record definitions. For example, here are excerpts from the DEFINE help module:

```
1 DEFINE
    You use the DEFINE commands to create and store definitions in the
    CDD for:

      o  Databases

      o  Dictionary directories

      o  Dictionaries

      o  Domains

      o  Ports

      o  Procedures

      o  Records

      o  Tables

    You use the DEFINE FILE command to create RMS data files for
    previously defined domains.

    You specify definitions in the CDD using path names.  For help on
    specifying path names see HELP Path_name.  For help on creating
    new versions of definitions see HELP REDEFINE.

2 DATABASE
    The DEFINE DATABASE command defines
         .
         .
         .
2 DICTIONARY
    The DEFINE DICTIONARY command creates a dictionary directory
         .
         .
         .
2 DOMAIN
    The DEFINE DOMAIN command stores a domain definition
         .
         .
         .
3 DBMS
    Stores in the CDD a definition of a DBMS domain that specifies a
    DBMS record type and database instance
         .
         .
         .
3 Network
    Stores a definition of a network domain in the CDD that points to
    data from another system linked to yours by DECnet
         .
         .
         .
```

Each numbered word in the file is a key. The top-level key is the keyword DEFINE. When you enter the command HELP DEFINE in response to the DTR> prompt, DATATRIEVE displays the keyword DEFINE and text that explains its use. In addition, DATATRIEVE displays the subtopics you can get further help on. The DEFINE subtopics are the second-level keys, that is, those words preceded by the number 2.

The second-level keys such as DATABASE, DICTIONARY, and DOMAIN also have text to explain them. When you use the command HELP DEFINE DOMAIN, DATATRIEVE displays the help text for DEFINE DOMAIN and the subtopics of DEFINE DOMAIN.

The subtopics of DEFINE DOMAIN are the third-level keys: RMS, DBMS, Network, Rdb, and View. If you use the command HELP DEFINE DOMAIN Network, DATATRIEVE displays the help text for defining network domains. DATATRIEVE does not display any subtopics because there are no fourth-level subtopics for DEFINE DOMAIN Network.

Figure 8-1 illustrates the structure of the DEFINE help file, the text for part of which is shown in the previous example.

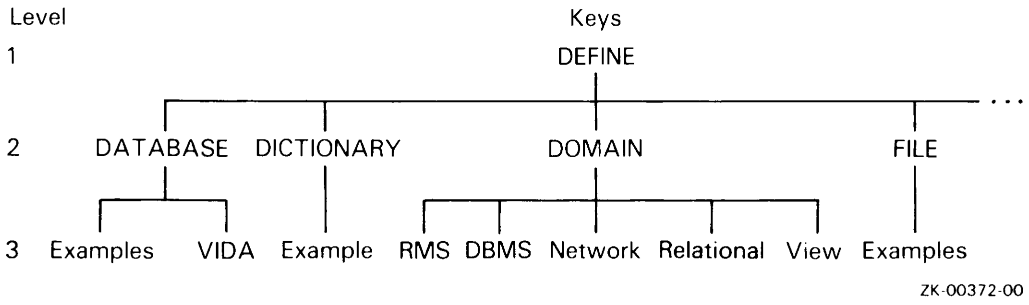

Figure 8-1: Structure of the DEFINE Help File

8.1.3.2 Changing a Help Text File — An example of a help file you can change is the one for DATATRIEVE functions. The help file for functions has one top-level key, followed by a help text:

```
1 Functions
    VAX DATATRIEVE has predefined functions you can use in interactive
    or callable DATATRIEVE.  You can use these functions to form value
    expressions or to set parameters for a process.
         .
         .
         .
```

Suppose you want to reorganize the functions help module. You want to categorize DATATRIEVE functions and to describe functions within each category. You also want to provide descriptions of user-defined functions.

Figure 8-2 shows one way you can structure your new help file.

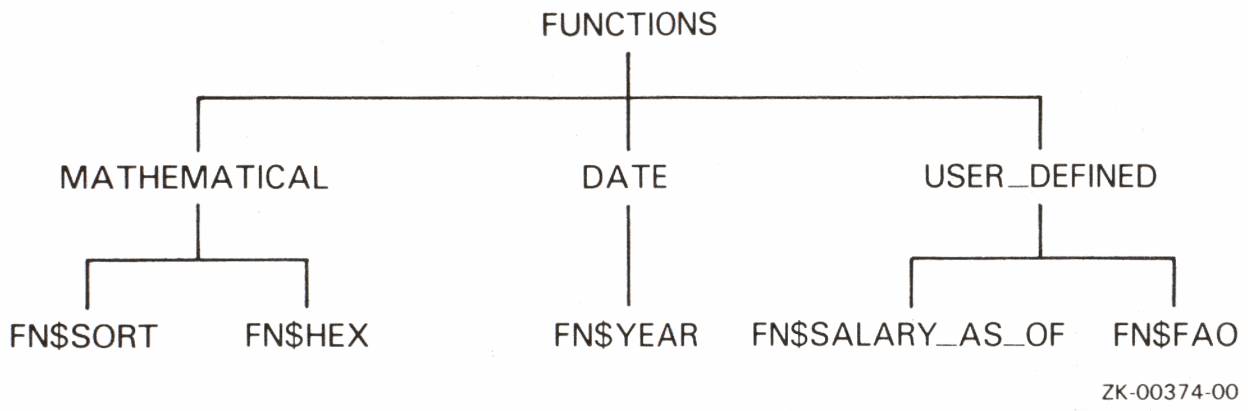

Figure 8-2: Structure of a Sample Help File

Help text you modify or create must have a hierarchical structure for DATATRIEVE to display it in the proper order. To understand this hierarchical help text structure, examine the organization of the .HLP files you extract from DTRHELP.HLB. For further explanation of how to design help text, refer to the VMS documentation set.

8.1.4 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 DEFINE module in DTRHELP.HLB with the changed text in DEFINE.HLP:

```
$ LIBRARY/HELP/REPLACE
_Library:       DTRHELP
_File:          DEFINE.HLP
```

8.1.5 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.

8.2 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.

Customizing DATATRIEVE Messages 9

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, VAX DBMS, and VAX EDT.

9.1 DATATRIEVE Messages

Each time you give DATATRIEVE a command or statement to execute, DATATRIEVE determines what message it should display. For each message, there is 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 hexidecimal 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.

A list describing the severity level, code, name, and text for all DATATRIEVE messages is supplied online as part of the DATATRIEVE installation procedure. See Appendix B for information on how to access the online message documentation.

9.1.1 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.

9.1.2 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)

                        FIRST       LAST              START            SUP
 ID      STATUS         NAME        NAME     DEPT     DATE      SALARY   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

DTR>
```

Table 9-1 lists the severity level, code, name, and text for each message in this example:

Table 9-1: Sample DATATRIEVE Messages

Example Number	Severity Level	Code	Name	Text
1	Error	008D8192	DTR$_ELTNOTDIC	Element <...> not found in dictionary.
2	Informational	008D857B	DTR$_LOOKINFOR	[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.

9.2 How to Change DATATRIEVE Messages

To change DATATRIEVE messages, you must:

1. Copy the DATATRIEVE message source file DTRMSGS.MSG 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 DTRMSGS.EXE.
5. Replace the old DTRMSGS.EXE file with the new one.

The following sections describe each of these steps.

9.2.1 Copy the Message Source File

The message source file DTRMSGS.MSG is located in DTR$LIBRARY. Use the COPY command to copy this file to one of your own directories. For example:

```
$ COPY DTR$LIBRARY:DTRMSGS.MSG DB3:[CASSIDY.WORK]DTRMSGS.MSG
```

This command creates a copy of DTRMSGS.MSG in the DB3:[CASSIDY.WORK] directory.

9.2.2 Edit the Message Source File

After you copy DTRMSGS.MSG, use an editor, such as EDT, to make the desired changes to the message text. The following sections describe the structure of DTRMSGS.MSG and explain how to edit it.

9.2.2.1 The Structure of the Message Source File — DTRMSGS.MSG, 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 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 FAO arguments to be included in the message. See the section entitled Using FAO Directives to Format Message Text later in this chapter 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 the section entitled Using FAO Directives to Format Message Text later in this chapter for more information on FAO directives.

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.

9.2.2.2 Changing the Message Text — An example of a message you may want to change is:

```
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 DTRMSGS.MSG is:

```
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 DTRMSGS.MSG 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 DTRMSGS.MSG 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.

9.2.2.3 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 9-2 summarizes the FAO directives used in DATATRIEVE message texts and lists the number of parameters required by each directive.

Table 9-2: 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
Output String Formatting		
!/	Inserts a new line (<CR> and <LF>).	None
!_	Inserts a tab.	
!	Inserts a form feed.	
!!	Inserts an exclamation point.	
!%S	Inserts the letter S if the most recently converted numeric value is not 1.	
!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 " ".	
!n*c	Repeats the specified character c, n times.	
Parameter Interpretation		
!−	Reuses the last parameter in the list.	None
!+	Skips the next parameter in the list.	

The following sections describe how to use FAO directives.

9.2.2.3.1 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$_LOOKINFOR. The definition for this message is:

```
LOOKINFOR/FAO=1/INFORMATIONAL-
<[Looking for !AC] >
```

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 11 for information about modifying DATATRIEVE text.

The second message in the example is DTR$_RECFOUND. The message definition for DTR$_RECFOUND is:

```
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 the section entitled Parameter Interpretation Directives later in this chapter. However, 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.

9.2.2.3.2 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.

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 DTRMSGS.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.!/ >
```

9.2.2.3.3 Parameter Interpretation Directives — You can use parameter interpretation directives to change the order of substitution directives. For example, here is a message that contains two string substitution directives:

```
DTR> PRIMT EMPLOYEES WITH LOCATION = "MK"
PRIMT EMPLOYEES WITH LOCATION = "MK"
^
Expected statement, encountered "PRIMT".
DTR>
```

The mistyped word in this example causes the error DTR$_SYNTAX. The message definition for DTR$_SYNTAX is:

```
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> PRIMT EMPLOYEES WITH LOCATION = "MK"
PRIMT EMPLOYEES WITH LOCATION = "MK"
^
You entered "PRIMT".
I was expecting a statement... Try again.
DTR>
```

To make this change, use an editor to edit the current message text in DTRMSGS.MSG:

```
<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 (...).

9.2.3 Compile the Message Source File

After you edit DTRMSGS.MSG to make the desired changes, use the MESSAGE command to compile the message source file:

```
$ MESSAGE DTRMSGS.MSG
```

This command creates a DTRMSGS.OBJ file in your default directory.

For more information about the MESSAGE command, enter HELP MESSAGE or refer to the VMS documentation set.

9.2.4 Link the Object File

After you create an object file, use the LINK command to create the file DTRMSGS.EXE:

```
$ LINK/SHAREABLE=DTRMSGS.EXE DTRMSGS.OBJ
```

The DTRMSGS.EXE file that the LINK command creates is a nonexecutable message file.

9.2.5 Replace the Old DTRMSGS.EXE File with the New One

If you want DATATRIEVE to display the changed messages to all users on your system, copy the new DTRMSGS.EXE message file to SYS$MESSAGE. You must have write access to SYS$MESSAGE to do this. Use the command:

```
$ COPY DTRMSGS.EXE SYS$MESSAGE:DTRMSGS.EXE
```

If you want DATATRIEVE to display the changed messages only to selected users, assign the process logical DTRMSGS to the new DTRMSGS.EXE file for each of these users. For example, if the new DTRMSGS.EXE is in the DB3:[CASSIDY.WORK] directory, use the following command:

```
$ ASSIGN "DB3:[CASSIDY.WORK]DTRMSGS.EXE" DTRMSGS
```

Users who want DATATRIEVE to display the changed messages can include this command in their LOGIN.COM file.

Customizing ADT and Guide Mode 10

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 CDD.

To use ADT, enter the following command at the DATATRIEVE prompt:

```
DTR> ADT
```

Guide Mode is a DATATRIEVE utility that guides users through a DATATRIEVE session with a series of prompts. In Guide Mode, the user 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.

10.1 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.

10.1.1 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
```

10.1.2 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.

10.1.2.1 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
```

The definition of ADT_TEXT is:

```
DOMAIN ADT_TEXT USING ADT_TEXT_REC ON DTRADT.DAT;
```

The record definition for ADT_TEXT is:

```
RECORD ADT_TEXT_REC USING
01      KEYED_TEXT.
        05  ACCESS_KEY  WORD.
        05  TEXT_STRING PIC X(80).
;
```

10.1.2.2 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:

```
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) :
```

Suppose you want to modify this prompt. Here is one way to do this:

```
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
```

(continued on next page)

```
                                                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 DATATRIEVE messages. For information about using FAO directives, refer to Chapter 9 of this manual and to the VMS documentation set.

10.1.2.3 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:

```
DATATRIEVE uses this file name in the DEFINE DOMAIN command
to find your data for the domain
```

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.

10.1.2.4 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 DTRMSGS.MSG. Messages specific to ADT begin with the letters ADT. If you want to modify these messages, follow the procedure described in Chapter 9.

10.1.3 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.

Note that DATATRIEVE supplies the default 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.

10.2 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.

The difference between these levels is the commands, statements, or options you can use. Table 10-1 shows what is available at the simple and advanced levels.

Table 10-1: 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

(continued on next page)

Table 10-1: Guide Mode Levels (Cont.)

Command, Statement, or Option	Level at which it is available
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:DTRMSGS.MSG. Prompt messages begin with GPR and informational messages begin with GUI. Chapter 9 explains how to customize these messages.

10.2.1 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
```

10.2.2 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)

  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 can not 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>
```

10.2.3 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
```

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.

Customizing DATATRIEVE Text 11

Chapters 8 through 10 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.

11.1 DATATRIEVE Text

There are several types of DATATRIEVE text you can modify:

- Syntax prompts
- Responses to SHOW commands
- Date text
- Default edit strings
- Statistical expressions
- Keywords

The following sections give examples of each type of text.

11.1.1 Syntax Prompt Text

Some DATATRIEVE text is embedded in the DATATRIEVE syntax prompt message, which is defined in the DATATRIEVE message file DTRMSGS.MSG. Here is an example:

```
DTR> READY
[Looking for dictionary path name] ------------------------------------(*)
CON> PERSONNEL
DTR> FIND
[Looking for name of domain, collection, or list] ---------------------(*)
CON> PERSONNEL
[24 records found]
DTR> SORT
[Looking for sort list] -----------------------------------------------(*)
CON> BY LAST_NAME
DTR>
```

Each of the highlighted messages is an instance of DTR$_LOOKINFOR, the DATATRIEVE syntax prompt. The DTR$_LOOKINFOR 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 9. 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".

11.1.2 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.

11.1.3 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.

11.1.4 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.

11.1.5 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.

11.1.6 Keywords

You can customize all DATATRIEVE keywords by using the DECLARE SYNONYM statement.

There are some keywords that you can also customize as DATATRIEVE text. These are:

ADT
CLOSE
EDIT
EXIT
GUIDE
HELP
HELP_PROMPT
HELP_WINDOW
HELP_LINES
NO_HELP_PROMPT
NO_HELP_WINDOW
OPEN
@ (command file invocation)

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.

11.2 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 DTRLIB.OLB.
5. Relink the DATATRIEVE shareable image.

The following sections describe each of these steps.

11.2.1 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.

11.2.2 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.

11.2.3 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.

11.2.4 Replace the Text Object File in DTRLIB.OLB

After you create the text object file, replace it in the library file DTR$LIBRARY:DTRLIB.OLB. If the new object file is in your default directory, you can use the command:

```
$ LIBRARY/REPLACE DTR$LIBRARY:DTRLIB DTRTEXT
```

11.2.5 Relink the DATATRIEVE Shareable Image

To relink VAX DATATRIEVE, execute the command file DTRBLD:

```
$ @DTR$LIBRARY:DTRBLD
```

Note that the files, DTRLIB.OLB and DTRBLD.COM, may have a suffix appended to the file name. DTRLIB.OLB might be DTRLIBFMS.OLB, for example.

Translating DATATRIEVE 12

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 V4.1
DEC Abfrage und Report System
Tippe HILFE fuer Hilfe
DTR> ZEIGE BEREICHE
Bereiche:
        BESITZER        FAMILIEN         JACHTEN         JAHRESBERICHT
        PERSONAL

DTR> BEREITE PERSONAL
DTR> ZEIGE BERIRT
Element "BERIRT" nicht im Lexikon gefunden.
DTR> ZEIGE BEREIT
Bereitete Bereiche:
   PERSONAL:  Bereich, RMS indiziert, geschuetzter Lesezugriff
           <CDD$TOP.KELLER.PERSONAL>
Keine Tabellen geladen.

DTR> ZEIGE FELDER FUER PERSONAL
PERSONAL
   PERSON
      AUSWEISNUMMER (NUMMER)    <Nummer, indizierter Schluesselfeld>
      STATUS    <Zeichenfolge>
      NAME (NAME)
         VORNAME        <Zeichenfolge>
         NACHNAME       <Zeichenfolge>
      ABTEILUNG (ABT)   <Zeichenfolge>
      ANFANGSDATUM (DATUM)      <Datum>
      GEHALT    <Nummer>
      AUFSEHER_NUMMER   <Nummer>
```

(continued on next page)

```
DTR> FINDE INGENIEURE IN PERSONAL MIT ABTEILUNG = "D98"
[4 Saetze gefunden]
DTR> DRUCKE NACHNAME, NUMMER, DATUM, GEHALT VON
[Suche Name eines Bereichs, einer Sammlung, oder Liste]
CON> INGENIEURE SORTIERT PER DATUM DANN
[Suche Anweisung]
CON> DRUCKE SPALTE 30, TOTAL GEHALT ("GESAMTGEHALT") BRAUCHEND
[Suche Masken- oder Aufbereitungszeichenkette]
CON> 999B999B"DM"

           AUSWEIS   ANFANGS
 NACHNAME  NUMMER     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> AUSGANG
$
```

As the example shows, you can make DATATRIEVE understand and respond in a foreign language. This chapter explains how.

12.1 Planning Your Translation

Before you begin your translation, you should read Chapters 7 through 11 of this manual 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.

Other text DATATRIEVE displays comes from the VAX Common Data Dictionary (CDD), VAX Record Management Services (RMS), VAX DBMS, and other VAX software products. This chapter discusses only the translation of DATATRIEVE.

12.2 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.

12.2.1 Translate 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:

```
DECLARE SYNONYM ALLES    FOR ALL,
                ALLE     FOR ALL,
                BEREICH  FOR DOMAIN,
                BEREICHE FOR DOMAINS,
                BEREITE  FOR READY,
                BEREIT   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> BEREITE PERSONNEL

DTR> ZEIGE BEREIT
```

Other DATATRIEVE keywords require multiple translations. For example, the keyword ALL requires two translations in German and three in Spanish:

English	German	Spanish
SHOW ALL	ZEIGE ALLES	MUESTRA TODO
PRINT ALL EMPLOYEES	DRUCKE ALLE ANGESTELLTE	IMPRIME TODOS EMPLEADOS
		IMPRIME TODAS EMPLEADAS

12.2.2 Translate 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.

3. Translate the text file NEWHELP.HLP. For example, the help text for the statement FIND is:

   ```
   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. Here is a sample translation:

   ```
   1 FINDE
       Die FINDE Anweisung enthaelt eine Satzsammlung von einem
       Bereich oder von einer vorheraufgestellten Sammlung.
       Das Format der Anweisung ist:

                              FINDE sa

       worin "sa" ein Satzwahl Ausdruck ist (siehe HILFE SA). Die
       Zahl der gefundenen Saetze ist auf Ihren Bildschirm angezeigt.

       Zum Beispiel, stelle eine Sammlung von JACHTEN auf:

            FINDE JACHTEN MIT LAENGE ZWISCHEN 26 UND 30

       Oder, stelle eine Sammlung genannt BILLIGE auf:

            FINDE 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
     ```

   - 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 FINDE, DATATRIEVE displays your translated help text.

For more detailed information about modifying DATATRIEVE help text, refer to Chapter 8.

12.2.3 Translate Messages

These are the steps necessary to translate DATATRIEVE messages:

1. Copy the DATATRIEVE message source file to one of your own directories:

   ```
   $ COPY DTR$LIBRARY:DTRMSGS.MSG
   _To:           DB2:[KELLER.MSG]NEWMSGS.MSG
   ```

   This command creates a copy of DTRMSGS.MSG called NEWMSGS.MSG 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   < - Zuruecktreten   PF2 - Bildschirm Hilfe >
   ```

3. Compile the translated message source file to create an object file:

```
$ MESSAGE DB2:[KELLER.MSG]NEWMSGS.MSG
```

4. Link the object file to create the file NEWMSGS.EXE:

```
$ LINK/EXECUTABLE=NEWMSGS.EXE NEWMSGS.OBJ
```

5. Replace the old DTRMSGS.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 DTRMSGS to NEWMSGS.EXE:

     ```
     $ ASSIGN "DB2:[KELLER.MSG]NEWMSGS.EXE" DTRMSGS
     ```

   - To update the message file for all users on the system, copy the NEWMSGS.EXE to DTRMSGS.EXE in the SYS$MESSAGE directory. (You must have write access to SYS$MESSAGE to do this.)

     ```
     $ COPY DB2:[KELLER.MSG]NEWMSGS.EXE
     _To:          SYS$MESSAGE:DTRMSGS.EXE
     ```

For more detailed information about modifying DATATRIEVE messages, refer to Chapter 9.

12.2.4 Translate 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 10.

12.2.5 Translate 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 DTRLIB.OLB:

   ```
   $ LIBRARY/REPLACE DTR$LIBRARY:DTRLIB.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 11.

12.2.6 Translate the Names of Functions

There are a number of DATATRIEVE functions you can use 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:DTRFND.MAR
_To:          DB2:[KELLER.FUN]DTRFND.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">
```

Your translated function definition is:

```
$DTR$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 DTRFND.MAR, assemble it with the command:

```
$ MACRO DB2:[KELLER.FUN]DTRFND.MAR
```

This command creates the object file DTRFND.OBJ.

4. Replace DTRFND.OBJ in the function library DTR$LIBRARY:DTRFUN. Use this command:

```
$ LIBRARY/REPLACE DTR$LIBRARY:DTRFUN.OLB
_File:          DB2:[KELLER.FUN]DTRFND.OBJ
```

You should now relink the DATATRIEVE shareable image.

12.2.7 Relink the DATATRIEVE Shareable Image

To relink DATATRIEVE, execute the command file DTR$LIBRARY:DTRBLD.COM:

```
$ @DTR$LIBRARY:DTRBLD
```

Note that the files DTRFND.MAR, DTRFUN.OLB, DTRLIB.OLB, and DTRBLD.COM may have a suffix appended to the file name. For example, DTRBLD.COM might be DTRBLDFMS.COM.

Definitions of the DATATRIEVE Access Block A

This appendix contains the definitions of the DATATRIEVE Access Block (DAB) in VAX FORTRAN, VAX COBOL, VAX BASIC, VAX PASCAL, and VAX PL/I. Copies of these inclusion files are supplied with the DATATRIEVE installation kit and are placed in the DTR$LIBRARY directory.

This appendix also 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 DATATRIEVE Access Block in VAX FORTRAN

```
c
c FORTRAN DATATRIEVE Access Block
c

        LOGICAL*1 DAB$B_BID, DAB$B_BLN, DAB$B_VER_LETTER, DAB(100)

        INTEGER*2 DAB$W_MSG_BUF_LEN, DAB$W_MSG_LEN,
        1         DAB$W_AUX_BUF_LEN, DAB$W_AUX_LEN,
        2         DAB$W_IDI, DAB$W_STATE, DAB$W_REC_LENGTH,
        3         DAB$W_VERSION, DAB$W_LEVEL, DAB$W_BASE_LEVEL,
        4         DAB$W_UDK_INDEX, DAB$W_COLUMNS_PAGE,
        5         DAB$W_TT_CHANNEL, DAB$W_CTLC_CHANNEL

        INTEGER*4 DAB$A_MSG_BUF, DAB$A_AUX_BUF,
        1         DAB$L_CONDITION, DAB$L_FLAGS, DAB$L_OPTIONS,
        2         DAB$L_KEYTABLE_ID, DAB$L_COMMAND_KEYBOARD,
        3         DAB$L_PROMPT_KEYBOARD
```

(continued on next page)

```
COMMON /DAB_COMMON/
1       DAB$B_BID,
2       DAB$B_BLN,
3       DAB$L_CONDITION,
4       DAB$A_MSG_BUF,
5       DAB$W_MSG_BUF_LEN,
6       DAB$W_MSG_LEN,
7       DAB$A_AUX_BUF,
8       DAB$W_AUX_BUF_LEN,
9       DAB$W_AUX_LEN,
1       DAB$W_IDI,
2       DAB$W_STATE,
3       DAB$L_FLAGS,
4       DAB$L_OPTIONS,
5       DAB$W_REC_LENGTH,
6       DAB$W_VERSION,
7       DAB$W_LEVEL,
8       DAB$B_VER_LETTER,
9       DAB$W_BASE_LEVEL,
1       DAB$W_UDK_INDEX,
2       DAB$W_COLUMNS_PAGE,
3       DAB$W_TT_CHANNEL,
4       DAB$W_CTLC_CHANNEL,
5       DAB$L_KEYTABLE_ID,
6       DAB$L_COMMAND_KEYBOARD,
7       DAB$L_PROMPT_KEYBOARD

EQUIVALENCE (DAB, DAB$B_BID)


INTEGER    DTR$K_STL_CMD, DTR$K_STL_PRMPT, DTR$K_STL_LINE,
1          DTR$K_STL_MSG, DTR$K_STL_PGET, DTR$K_STL_PPUT,
2          DTR$K_STL_CONT, DTR$K_STL_UDK, DTR$K_STL_END_UDK


PARAMETER (DTR$K_STL_CMD=1,
1          DTR$K_STL_PRMPT=2,
2          DTR$K_STL_LINE=3,
3          DTR$K_STL_MSG=4,
4          DTR$K_STL_PGET=5,
5          DTR$K_STL_PPUT=6,
6          DTR$K_STL_CONT=7,
7          DTR$K_STL_UDK=8,
8          DTR$K_STL_END_UDK=9)

INTEGER DTR$K_SEMI_COLON_OPT, DTR$K_UNQUOTED_LIT,
1       DTR$K_SYNTAX_PROMPT, DTR$K_IMMED_RETURN,
2       DTR$K_FORMS_ENABLE, DTR$K_VERIFY, DTR$K_CONTEXT_SEARCH,
3       DTR$K_HYPHEN_DISABLED, DTR$K_MORE_COMMANDS, DTR$K_ABORT,
4       DTR$K_LOCK_WAIT
```

```
PARAMETER (DTR$K_SEMI_COLON_OPT=1,
1          DTR$K_UNQUOTED_LIT=16,
2          DTR$K_SYNTAX_PROMPT=32,
3          DTR$K_IMMED_RETURN=64,
4          DTR$K_FORMS_ENABLE=128,
5          DTR$K_VERIFY=256,
6          DTR$K_CONTEXT_SEARCH=2048,
7          DTR$K_HYPHEN_DISABLED=4096,
8          DTR$K_MORE_COMMANDS=8192,
9          DTR$K_ABORT=16384,
1          DTR$K_LOCK_WAIT=32768)

INTEGER    DTR$M_OPT_CMD, DTR$M_OPT_PRMPT,   DTR$M_OPT_LINE,
1          DTR$M_OPT_MSG, DTR$M_OPT_PGET, DTR$M_OPT_PPUT,
2          DTR$M_OPT_CONT, DTR$M_OPT_UDK, DTR$M_OPT_DTR_UDK,
3          DTR$M_OPT_END_UDK, DTR$M_OPT_UNWIND,
4          DTR$M_OPT_CONTROL_C, DTR$M_OPT_STARTUP,
5          DTR$M_OPT_FOREIGN, DTR$M_OPT_BANNER, DTR$M_OPT_REMOVE_CTLC,
6          DTR$M_OPT_KEYDEFS

PARAMETER (DTR$M_OPT_CMD=1,
1          DTR$M_OPT_PRMPT=2,
2          DTR$M_OPT_LINE=4,
3          DTR$M_OPT_MSG=8,
4          DTR$M_OPT_PGET=16,
5          DTR$M_OPT_PPUT=32,
6          DTR$M_OPT_CONT=64,
7          DTR$M_OPT_UDK=128,
8          DTR$M_OPT_DTR_UDK=256,
9          DTR$M_OPT_END_UDK=512,
1          DTR$M_OPT_UNWIND=1024,
2          DTR$M_OPT_CONTROL_C=2048,
3          DTR$M_OPT_STARTUP=4096,
4          DTR$M_OPT_FOREIGN=8192,
5          DTR$M_OPT_BANNER=16384,
6          DTR$M_OPT_REMOVE_CTLC=32768,
7          DTR$M_OPT_KEYDEFS=65536)

INTEGER    DTR$K_UDK_SET, DTR$K_UDK_SET_NO, DTR$K_UDK_SHOW,
1          DTR$K_UDK_STATEMENT,  DTR$K_UDK_COMMAND

PARAMETER (DTR$K_UDK_SET=1,
1          DTR$K_UDK_SET_NO=2,
2          DTR$K_UDK_SHOW=3,
3          DTR$K_UDK_STATEMENT=4,
4          DTR$K_UDK_COMMAND=5)

INTEGER    DTR$K_TOK_TOKEN, DTR$K_TOK_PICTURE,
1          DTR$K_TOK_FILENAME, DTR$K_TOK_COMMAND,
2          DTR$K_TOK_TEST_TOKEN, DTR$K_TOK_LIST_ELEMENT,
3          DTR$K_TOK_TEST_EOL
```

(continued on next page)

```
        PARAMETER (DTR$K_TOK_TOKEN=1,
        1          DTR$K_TOK_PICTURE=2,
        2          DTR$K_TOK_FILENAME=3,
        3          DTR$K_TOK_COMMAND=4,
        4          DTR$K_TOK_TEST_TOKEN=5,
        5          DTR$K_TOK_LIST_ELEMENT=6,
        6          DTR$K_TOK_TEST_EOL=7)


        CHARACTER*80  MSG_BUFF
        CHARACTER*20  AUX_BUFF
        COMMON /DTR$BUFFERS/MSG_BUFF,AUX_BUFF

        EXTERNAL   DTR$_SUCCESS
```

A.2 DATATRIEVE Access Block in VAX COBOL

```
* COBOL DATATRIEVE Access Block

01      DAB.
        03 BLOCK_ID.
                05 DAB$B_BID         PIC X.
                05 DAB$B_BLN         PIC X.
        03 DAB$L_CONDITION           PIC S9(9) COMP.
        03 MSG_BUFFER.
                05 DAB$A_MSG_BUF     PIC 9(9) COMP.
                05 DAB$W_MSG_BUF_LEN PIC 9(4) COMP.
                05 DAB$W_MSG_LEN     PIC 9(4) COMP.
        03 AUX_BUFFER.
                05 DAB$A_AUX_BUF     PIC 9(9) COMP.
                05 DAB$W_AUX_BUF_LEN PIC 9(4) COMP.
                05 DAB$W_AUX_LEN     PIC 9(4) COMP.
        03 DAB$W_IDI                 PIC 9(4) COMP.
        03 DAB$W_STATE               PIC 9(4) COMP.
           88 DTR$K_STL_CMD          VALUE 1.
           88 DTR$K_STL_PRMPT        VALUE 2.
           88 DTR$K_STL_LINE         VALUE 3.
           88 DTR$K_STL_MSG          VALUE 4.
           88 DTR$K_STL_PGET         VALUE 5.
           88 DTR$K_STL_PPUT         VALUE 6.
           88 DTR$K_STL_CONT         VALUE 7.
           88 DTR$K_STL_UDK          VALUE 8.
           88 DTR$K_STL_END_UDK      VALUE 9.
        03 DAB$L_FLAGS               PIC 9(9) COMP.
        03 DAB$L_OPTIONS             PIC 9(9) COMP.
        03 DAB$W_REC_LEN             PIC 9(4) COMP.
        03 DAB$W_VERSION             PIC 9(4) COMP.
        03 DAB$W_LEVEL               PIC 9(4) COMP.
        03 DAB$B_VER_LETTER          PIC X.
        03 DAB$W_BASE_LEVEL          PIC 9(4) COMP.
        03 DAB$W_UDK_INDEX           PIC 9(4) COMP.
        03 DAB$W_COLUMNS_PAGE        PIC 9(4) COMP.
        03 DAB$W_TT_CHANNEL          PIC 9(4) COMP.
        03 DAB$W_CTLC_CHANNEL        PIC 9(4) COMP.
        03 DAB$L_KEYTABLE_ID         PIC 9(9) COMP.
        03 DAB$L_COMMAND_KEYBOARD    PIC 9(9) COMP.
        03 DAB$L_PROMPT_KEYBOARD     PIC 9(9) COMP.
        03 DAB$REST_OF_DAB           PIC X(37).
```

```
01 INITIAL_OPTIONS.
    03 DTR$K_SEMI_COLON_OPT      PIC 9(9) COMP VALUE 1.
    03 DTR$K_UNQUOTED_LIT        PIC 9(9) COMP VALUE 16.
    03 DTR$K_SYNTAX_PROMPT       PIC 9(9) COMP VALUE 32.
    03 DTR$K_IMMED_RETURN        PIC 9(9) COMP VALUE 64.
    03 DTR$K_FORMS_ENABLE        PIC 9(9) COMP VALUE 128.
    03 DTR$K_VERIFY              PIC 9(9) COMP VALUE 256.
    03 DTR$K_CONTEXT_SEARCH      PIC 9(9) COMP VALUE 2048.
    03 DTR$K_HYPHEN_DISABLED     PIC 9(9) COMP VALUE 4096.
    03 DTR$K_MORE_COMMANDS       PIC 9(9) COMP VALUE 8192.
    03 DTR$K_ABORT               PIC 9(9) COMP VALUE 16384.
    03 DTR$K_LOCK_WAIT           PIC 9(9) COMP VALUE 32768.

01 DTR$DTR_OPTIONS.
    03 DTR$M_OPT_CMD             PIC 9(9) COMP VALUE  1.
    03 DTR$M_OPT_PRMPT           PIC 9(9) COMP VALUE  2.
    03 DTR$M_OPT_LINE            PIC 9(9) COMP VALUE  4.
    03 DTR$M_OPT_MSG             PIC 9(9) COMP VALUE  8.
    03 DTR$M_OPT_PGET            PIC 9(9) COMP VALUE  16.
    03 DTR$M_OPT_PPUT            PIC 9(9) COMP VALUE  32.
    03 DTR$M_OPT_CONT            PIC 9(9) COMP VALUE  64.
    03 DTR$M_OPT_UDK             PIC 9(9) COMP VALUE  128.
    03 DTR$M_OPT_DTR_UDK         PIC 9(9) COMP VALUE  256.
    03 DTR$M_OPT_END_UDK         PIC 9(9) COMP VALUE  512.
    03 DTR$M_OPT_UNWIND          PIC 9(9) COMP VALUE  1024.
    03 DTR$M_OPT_CONTROL_C       PIC 9(9) COMP VALUE  2048.
    03 DTR$M_OPT_STARTUP         PIC 9(9) COMP VALUE  4096.
    03 DTR$M_OPT_FOREIGN         PIC 9(9) COMP VALUE  8192.
    03 DTR$M_OPT_BANNER          PIC 9(9) COMP VALUE  16384.
    03 DTR$M_OPT_REMOVE_CTLC     PIC 9(9) COMP VALUE  32768.
    03 DTR$M_OPT_KEYDEFS         PIC 9(9) COMP VALUE  65536.

01 DTR$UDK_CONTEXTS.
    03 DTR$K_UDK_SET             PIC 9(4) COMP VALUE 1.
    03 DTR$K_UDK_SET_NO          PIC 9(4) COMP VALUE 2.
    03 DTR$K_UDK_SHOW            PIC 9(4) COMP VALUE 3.
    03 DTR$K_UDK_STATEMENT       PIC 9(4) COMP VALUE 4.
    03 DTR$K_UDK_COMMAND         PIC 9(4) COMP VALUE 5.

01 DTR$UDK_TOKENS.
    03 DTR$K_TOK_TOKEN           PIC 9(4) COMP VALUE 1.
    03 DTR$K_TOK_PICTURE         PIC 9(4) COMP VALUE 2.
    03 DTR$K_TOK_FILENAME        PIC 9(4) COMP VALUE 3.
    03 DTR$K_TOK_COMMAND         PIC 9(4) COMP VALUE 4.
    03 DTR$K_TOK_TEST_TOKEN      PIC 9(4) COMP VALUE 5.
    03 DTR$K_TOK_LIST_ELEMENT    PIC 9(4) COMP VALUE 6.
    03 DTR$K_TOK_TEST_EOL        PIC 9(4) COMP VALUE 7.

01 MSG_BUFF                      PIC X(80).
01 AUX_BUFF                      PIC X(20).

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

A.3 DATATRIEVE Access Block in VAX BASIC

```
MAP (ACCESSBLOCK) &
BYTE DAB$B_BID, &
BYTE DAB$B_BLN, &
LONG DAB$L_CONDITION, &
LONG DAB$A_MSG_BUF, &
WORD DAB$W_MSG_BUF_LEN, &
WORD DAB$W_MSG_LEN, &
LONG DAB$A_AUX_BUF, &
WORD DAB$W_AUX_BUF_LEN, &
WORD DAB$W_AUX_LEN, &
WORD DAB$W_IDI, &
WORD DAB$W_STATE, &
LONG DAB$L_FLAGS, &
LONG DAB$L_OPTIONS, &
WORD DAB$W_REC_LENGTH, &
WORD DAB$W_VERSION, &
WORD DAB$W_LEVEL, &
BYTE DAB$B_VER_LETTER, &
WORD DAB$W_BASE_LEVEL, &
WORD DAB$W_UDK_INDEX, &
WORD DAB$W_COLUMNS_PAGE, &
WORD DAB$W_TT_CHANNEL, &
WORD DAB$W_CTLC_CHANNEL, &
LONG DAB$L_KEYTABLE_ID, &
LONG DAB$L_COMMAND_KEYBOARD, &
LONG DAB$L_PROMPT_KEYBOARD

MAP (ACCESSBLOCK) STRING DAB = 100

DECLARE INTEGER CONSTANT DTR$K_STL_CMD = 1% ,&
                         DTR$K_STL_PRMPT = 2%,&
                         DTR$K_STL_LINE = 3%,&
                         DTR$K_STL_MSG = 4%,&
                         DTR$K_STL_PGET = 5%,&
                         DTR$K_STL_PPUT = 6%,&
                         DTR$K_STL_CONT = 7%,&
                         DTR$K_STL_UDK = 8%,&
                         DTR$K_STL_END_UDK = 9%

DECLARE INTEGER CONSTANT DTR$K_SEMI_COLON_OPT = 1%, &
                         DTR$K_UNQUOTED_LIT = 16%, &
                         DTR$K_SYNTAX_PROMPT = 32%, &
                         DTR$K_IMMED_RETURN = 64%, &
                         DTR$K_FORMS_ENABLE = 128%, &
                         DTR$K_VERIFY = 256%, &
                         DTR$K_CONTEXT_SEARCH = 2048%, &
                         DTR$K_HYPHEN_DISABLED = 4096%, &
                         DTR$K_MORE_COMMANDS = 8192%, &
                         DTR$K_ABORT = 16384%, &
                         DTR$K_LOCK_WAIT = 32768%
```

```
DECLARE INTEGER CONSTANT DTR$M_OPT_CMD = 1%, &
                         DTR$M_OPT_PRMPT = 2%, &
                         DTR$M_OPT_LINE = 4%, &
                         DTR$M_OPT_MSG = 8%, &
                         DTR$M_OPT_PGET = 16%, &
                         DTR$M_OPT_PPUT = 32%, &
                         DTR$M_OPT_CONT = 64%, &
                         DTR$M_OPT_UDK = 128%, &
                         DTR$M_OPT_DTR_UDK = 256%, &
                         DTR$M_OPT_END_UDK = 512%, &
                         DTR$M_OPT_UNWIND = 1024%, &
                         DTR$M_OPT_CONTROL_C = 2048%, &
                         DTR$M_OPT_STARTUP = 4096%, &
                         DTR$M_OPT_FOREIGN = 8192%, &
                         DTR$M_OPT_BANNER = 16384%, &
                         DTR$M_OPT_REMOVE_CTLC = 32768%, &
                         DTR$M_OPT_KEYDEFS = 64536%

DECLARE INTEGER CONSTANT DTR$K_UDK_SET = 1%, &
                         DTR$K_UDK_SET_NO = 2%, &
                         DTR$K_UDK_SHOW = 3%, &
                         DTR$K_UDK_STATEMENT = 4%, &
                         DTR$K_UDK_COMMAND = 5%

DECLARE INTEGER CONSTANT DTR$K_TOK_TOKEN = 1%, &
                         DTR$K_TOK_PICTURE = 2%, &
                         DTR$K_TOK_FILENAME = 3%, &
                         DTR$K_TOK_COMMAND = 4%, &
                         DTR$K_TOK_TEST_TOKEN = 5%, &
                         DTR$K_TOK_LIST_ELEMENT = 6%, &
                         DTR$K_TOK_TEST_EOL = 7%

COMMON(DTR$BUFFERS) STRING MSG_BUFF = 80, AUX_BUFF = 20

EXTERNAL INTEGER CONSTANT DTR$_SUCCESS
```

A.4 DATATRIEVE Access Block in VAX PASCAL

Several improvements were made to the DAB.PAS file in VAX DATATRIEVE Version 4.0. As a result of these improvements, you may have to modify PASCAL programs that were written using the DATATRIEVE Call Interface for prior versions of VAX DATATRIEVE.

In particular, the following variable declarations have been removed from DAB.PAS:

```
VAR

    REC_BUFF      : REC_TYPE;
    COMMAND       : VARYING [80] OF CHAR;
    STRING        : VARYING [80] OF CHAR;
    TEST_TOKEN    : VARYING [80] OF CHAR;
    PROMPT        : VARYING [80] OF CHAR;
    STK_SIZE      : WORD;
    INIT_OPTIONS  : WORD;
    DTR_OPTIONS   : WORD;
    DTR_STATUS    : INTEGER;
    TOKEN_LENGTH  : INTEGER;
    INDEX         : INTEGER;
    CONTEXT       : INTEGER;
    TOKEN_TYPE    : INTEGER;
```

These variable declarations were removed because they conflicted with names reserved in PASCAL or common variable names used in PASCAL programs.

If your program refers to the variables that have been removed, you will need to define the necessary variables explicitly within your program. See the sample declarations located at the end of DAB.PAS for suggestions to help you define the variables.

```
{ Pascal DATATRIEVE Access Block }

{ Constant declarations section }

CONST

{Stallpoints}

    DTR$K_STL_CMD=1;
    DTR$K_STL_PRMPT=2;
    DTR$K_STL_LINE=3;
    DTR$K_STL_MSG=4;
    DTR$K_STL_PGET=5;
    DTR$K_STL_PPUT=6;
    DTR$K_STL_CONT=7;
    DTR$K_STL_UDK=8;
    DTR$K_STL_END_UDK=9;
```

```
{Initialization Options}

    DTR$K_SEMI_COLON_OPT=1;
    DTR$K_UNQUOTED_LIT=16;
    DTR$K_SYNTAX_PROMPT=32;
    DTR$K_IMMED_RETURN=64;
    DTR$K_FORMS_ENABLE=128;
    DTR$K_VERIFY=256;
    DTR$K_CONTEXT_SEARCH=2048;
    DTR$K_HYPHEN_DISABLED=4096;
    DTR$K_MORE_COMMANDS=8192;
    DTR$K_ABORT=16384;
    DTR$K_LOCK_WAIT=32768;

{DTR$DTR call options }

    DTR$M_OPT_CMD=1;
    DTR$M_OPT_PRMPT=2;
    DTR$M_OPT_LINE=4;
    DTR$M_OPT_MSG=8;
    DTR$M_OPT_PGET=16;
    DTR$M_OPT_PPUT=32;
    DTR$M_OPT_CONT=64;
    DTR$M_OPT_UDK=128;
    DTR$M_OPT_DTR_UDK=256;
    DTR$M_OPT_END_UDK=512;
    DTR$M_OPT_UNWIND=1024;
    DTR$M_OPT_CONTROL_C=2048;
    DTR$M_OPT_STARTUP=4096;
    DTR$M_OPT_FOREIGN=8192;
    DTR$M_OPT_BANNER=16384;
    DTR$M_OPT_REMOVE_CTLC=32768;
    DTR$M_OPT_KEYDEFS=65536;

{User-defined keyword types}

    DTR$K_UDK_SET=1;
    DTR$K_UDK_SET_NO=2;
    DTR$K_UDK_SHOW=3;
    DTR$K_UDK_STATEMENT=4;
    DTR$K_UDK_COMMAND=5;

{User-defined keyword token types}

    DTR$K_TOK_TOKEN=1;
    DTR$K_TOK_PICTURE=2;
    DTR$K_TOK_FILENAME=3;
    DTR$K_TOK_COMMAND=4;
    DTR$K_TOK_TEST_TOKEN=5;
    DTR$K_TOK_LIST_ELEMENT=6;
    DTR$K_TOK_TEST_EOL=7;

{DTR Status messages}

    DTR$_SUCCESS=9274723;
```

(continued on next page)

```
{DTR$INFO type constants}
    DTR$K_INF_TYPE_DOMAIN=1;
    DTR$K_INF_TYPE_COLLECTION=2;
    DTR$K_INF_TYPE_KEYWORD=3;
    DTR$K_INF_TYPE_DIC_NAME=4;
    DTR$K_INF_TYPE_GLV=5;
    DTR$K_INF_TYPE_PLOT=6;

{DTR$LOOKUP info constants}
    DTR$K_INF_DOM_FLD=9;
    DTR$K_INF_DOM_FORM=10;
    DTR$K_INF_DOM_SHARE=11;
    DTR$K_INF_DOM_ACCESS=12;
    DTR$K_INF_DOM_NAME=13;
    DTR$K_INF_DOM_NEXT_DOM=14;
    DTR$K_INF_DOM_SSC=15;
    DTR$K_INF_FLD_NAME=16;
    DTR$K_INF_FLD_QNAME=17;
    DTR$K_INF_FLD_PICTURE=18;
    DTR$K_INF_FLD_EDIT=19;
    DTR$K_INF_FLD_DTYPE=20;
    DTR$K_INF_FLD_OFFSET=21;
    DTR$K_INF_FLD_LENGTH=22;
    DTR$K_INF_FLD_SCALE=23;
    DTR$K_INF_FLD_CHILD=24;
    DTR$K_INF_FLD_CNT=25;
    DTR$K_INF_FLD_LIST=26;
    DTR$K_INF_FLD_REDEFINES=27;
    DTR$K_INF_FLD_VIRTUAL=28;
    DTR$K_INF_FLD_FILLER=29;
    DTR$K_INF_FLD_MISSING=30;
    DTR$K_INF_COL_CURSOR=1;
    DTR$K_INF_COL_SIZE=2;
    DTR$K_INF_COL_FLD=3;
    DTR$K_INF_COL_DROPPED=4;
    DTR$K_INF_COL_ERASED=5;
    DTR$K_INF_COL_INVISIBLE=6;
    DTR$K_INF_COL_NAME=7;
    DTR$K_INF_COL_NEXT_COL=8;
    DTR$K_INF_GLV_FIRST_DOM=34;
    DTR$K_INF_GLV_FIRST_COL=35;
    DTR$K_INF_GLV_FIRST_SSC=36;
    DTR$K_INF_FRM_NAME=32;
    DTR$K_INF_FRM_LIBRARY=33;
    DTR$K_INF_SSC_NAME=46;
    DTR$K_INF_SSC_SET=47;
    DTR$K_INF_SSC_NEXT=48;
    DTR$K_INF_SET_NAME=49;
    DTR$K_INF_SET_NEXT=50;
    DTR$K_INF_SET_SDP=51;
    DTR$K_INF_SET_SINGULAR=52;
    DTR$K_INF_SDP_NEXT=53;
    DTR$K_INF_SDP_DOMAIN=54;
    DTR$K_INF_SDP_TENANCY=55;
    DTR$K_INF_SDP_INSERT=56;
    DTR$K_INF_SDP_RETAIN=57;
    DTR$K_INF_FLD_QHDR=31;
    DTR$K_INF_HDR_CNT=40;
    DTR$K_INF_HDR_STRING=41;
    DTR$K_INF_GLV_STA_OBJ=37;
    DTR$K_INF_GLV_STA_CNT=38;
    DTR$K_INF_GLV_STA_LINE=39;
    DTR$K_INF_PLO_CNT=42;
    DTR$K_INF_PLO_PAI=43;
    DTR$K_INF_PAI_PROMPT=44;
    DTR$K_INF_PAI_DTYPE=45;
    DTR$K_INF_DOM_REC_LEN=58;

{DTR$LOOKUP answer constants}
    DTR$K_INF_DOM_ACCESS_READ=1;
    DTR$K_INF_DOM_ACCESS_WRITE=2;
    DTR$K_INF_DOM_ACCESS_MODIFY=3;
    DTR$K_INF_DOM_ACCESS_EXTEND=4;
    DTR$K_INF_DOM_SHARE_EXCLUSIVE=1;
    DTR$K_INF_DOM_SHARE_SHARED=2;
    DTR$K_INF_DOM_SHARE_PROTECT=3;

{ Type declarations section }

TYPE
    DTR$WORD = [WORD] 0..65535;

{Dab}

    DTR_access_block = packed record
        DAB$B_BID                 : char;
        DAB$B_BLN                 : char;
        DAB$L_CONDITION           : integer;
        DAB$A_MSG_BUF             : integer;
        DAB$W_MSG_BUF_LEN         : dtr$word;
        DAB$W_MSG_LEN             : dtr$word;
        DAB$A_AUX_BUF             : integer;
        DAB$W_AUX_BUF_LEN         : dtr$word;
        DAB$W_AUX_LEN             : dtr$word;
        DAB$W_IDI                 : dtr$word;
        DAB$W_STATE               : dtr$word;
        DAB$L_FLAGS               : integer;
        DAB$L_OPTIONS             : integer;
        DAB$W_REC_LENGTH          : dtr$word;
        DAB$W_VERSION             : dtr$word;
        DAB$W_LEVEL               : dtr$word;
        DAB$B_VER_LETTER          : char;
        DAB$W_BASE_LEVEL          : dtr$word;
        DAB$W_UDK_INDEX           : dtr$word;
        DAB$W_COLUMNS_PAGE        : dtr$word;
        DAB$W_TT_CHANNEL          : dtr$word;
        DAB$W_CTLC_CHANNEL        : dtr$word;
        DAB$L_KEYTABLE_ID         : integer;
        DAB$L_COMMAND_KEYBOARD    : integer;
        DAB$L_PROMPT_KEYBOARD     : integer;
        DAB$REST_OF_DAB           : PACKED ARRAY [1..37] OF CHAR;
    end;
```

(continued on next page)

```
    DTR$REC_TYPE = PACKED ARRAY [1..80] OF CHAR;

{ Variable declarations section }
VAR
        DAB      : DTR_ACCESS_BLOCK;
        MSG_BUFF: [volatile] PACKED ARRAY [1..80] OF CHAR;
        AUX_BUFF: [volatile] PACKED ARRAY [1..20] OF CHAR;

{ External procedure declarations }

function DTR$INIT (
        %ref dab : DTR_access_block;
        stk_size : [truncate] dtr$word := %immed 0;
        var msg_buff:[truncate,class_s,volatile]packed array [l1..u1:integer]
               of char:= %immed 0;
        var aux_buff:[truncate,class_s,volatile]packed array [l2..u2:integer]
               of char:= %immed 0;
        init_options : [truncate] dtr$word := %immed 0
        ) : integer; extern;

function DTR$DTR (
        %ref dab : DTR_access_block;
        dtr_options : integer
        ) : integer; extern;

function DTR$COMMAND (
        %ref dab : DTR_access_block;
        command_line: [class_s, list] packed array [l1..u1 : integer]  of char
               := %immed 0
        ) : integer; extern;

function DTR$CONTINUE ( %ref dab : DTR_access_block ) : integer; extern;

function DTR$PUT_VALUE (
        %ref dab : DTR_access_block;
        text_string : [truncate,class_s] packed array [l1..u1 : integer]
        of char := %immed 0
        ) : integer; extern;

function DTR$GET_PORT (
        %ref dab : DTR_access_block;
        %ref rec_buff : [unsafe] packed array [l1..u1 : integer] of char
        ) : integer; extern;

function DTR$PUT_PORT (
        %ref dab : DTR_access_block;
        %ref rec_buff : [unsafe] packed array [l1..u1 : integer] of char
        ) : integer; extern;

function DTR$PORT_EOF ( %ref dab : DTR_access_block ): integer; extern;

function DTR$UNWIND   ( %ref dab : DTR_access_block ) : integer; extern;

function DTR$CREATE_UDK (
        %ref dab : DTR_access_block;
        text_string : [class_s] packed array [l1..u1 : integer] of char;
        %ref udk_index : integer;
        context : integer
        ) : integer; extern;
```

```
function DTR$GET_STRING (
        %ref dab : DTR_access_block;
        token_type : integer;
        text_string : [class_s] packed array [l1..u1 : integer] of char;
        %ref token_length : [truncate] integer := %immed 0;
        test_token : [class_s,truncate] packed array [l2..u2:integer] of
        char :=%immed 0
        ) : integer; extern;

function DTR$END_UDK ( %ref dab : DTR_access_block ) : integer; extern;

function DTR$PUT_OUTPUT (
        %ref dab : DTR_access_block;
        text_string : [class_s] packed array [l1..u1 : integer] of char;
        prompt : [truncate,class_s] packed array [l2..u2 : integer]
              of char := %immed 0
        ) : integer; extern;

function DTR$PRINT_DAB ( %ref dab : DTR_access_block ) : integer; extern;

function DTR$LOOKUP (
        %ref dab : DTR_access_block;
        object_type : integer;
        %ref object_id : integer;
        object_name : [class_s,truncate] packed array [a.integer] of
        char := %immed 0
        ) : integer; external;

function DTR$INFO (
        %ref dab : DTR_access_block;
        object_id : integer;
        info_code : integer;
        %ref ret_val : integer;
        var text_string : [class_s,truncate] packed array [a.integer] of char
                := %immed 0;
        %immed info_index : [truncate] integer := %immed 0
        ) : integer; external;

function DTR$FINISH ( %ref dab : DTR_access_block ) : integer; extern;

{ Sample declarations of parameters to the external procedures declared above:

        REC_BUFF        : DTR$REC_TYPE;
        COMMAND_LINE    : VARYING [80] OF CHAR;
        TEXT_STRING     : VARYING [80] OF CHAR;
        TEST_TOKEN      : VARYING [80] OF CHAR;
        PROMPT          : VARYING [80] OF CHAR;
        STK_SIZE        : dtr$word;
        INIT_OPTIONS    : dtr$word;
        DTR_OPTIONS     : dtr$word;
        DTR_STATUS      : INTEGER;
        TOKEN_LENGTH    : INTEGER;
        INFO_INDEX      : INTEGER;
        UDK_INDEX       : INTEGER;
        CONTEXT         : INTEGER;
        TOKEN_TYPE      : INTEGER;                                  }
```

A.5 DATATRIEVE Access Block in VAX PL/I

```
/* PL/I DATATRIEVE Access Block declarations */
/* Last revision: 07-Jan-1986 KG */

/*
 *  PL/I Usage notes
 *
 *  The initialization options, which are documented as DTR$K_xxx constants,
 *  are named DTR$M_xxx for PL/I.  Both the initialization options and the
 *  DTR$DTR options are represented as bit strings, and thus can be combined
 *  with the bitwise or (|) operator.  For example:
 *
 *      STATUS = DTR$DTR( DAB, DTR$M_OPT_CMD | DTR$M_OPT_PRMPT );
 *
 *  The entry point definitions use several features that are only available
 *  with VAX PL/I V3.0 and later.  (In particular, the OPTIONAL, TRUNCATE,
 *  LIST and ANY DESCRIPTOR.)
 *
 *  If you want the typical variable declarations used in many of the examples,
 *  they can be declared as follows:
 *
 *  DCL 1 DAB LIKE DTR_ACCESS_BLOCK;
 *  DCL MSG_BUF CHAR(80);
 *  DCL AUX_BUF CHAR(20);
 *
 *  If for some reason, you want to use your own declaration of the DAB, but
 *  still want to use the entry point definitions in this file, you can use
 *  the REFERENCE built-in function to override the declaration.  For example:
 *
 *  %INCLUDE 'DTR$LIBRARY:DAB';
 *  DCL 1 MY_DAB,
 *        2 ...
 *  STATUS = DTR$DTR( REF(MY_DAB), DTR$M_OPT_CMD | DTR$M_OPT_PRMPT );
 *
 *  Note that the ANY CHARACTER(*) parameter type can accept either CHARACTER
 *  or CHARACTER VARYING strings without creating dummy arguments.
 */

/* Constant declarations section */

/* Stallpoints */
%replace DTR$K_STL_CMD by 1;
%replace DTR$K_STL_PRMPT by 2;
%replace DTR$K_STL_LINE by 3;
%replace DTR$K_STL_MSG by 4;
%replace DTR$K_STL_PGET by 5;
%replace DTR$K_STL_PPUT by 6;
%replace DTR$K_STL_CONT by 7;
%replace DTR$K_STL_UDK by 8;
%replace DTR$K_STL_END_UDK by 9;
```

```
/* Initialization Options */
%replace DTR$M_SEMI_COLON_OPT by '10000000000000000000000000000000'b;
%replace DTR$M_UNQUOTED_LIT by  '00001000000000000000000000000000'b;
%replace DTR$M_SYNTAX_PROMPT by '00000100000000000000000000000000'b;
%replace DTR$M_IMMED_RETURN by '00000010000000000000000000000000'b;
%replace DTR$M_FORMS_ENABLE by '00000001000000000000000000000000'b;
%replace DTR$M_VERIFY by '00000000100000000000000000000000'b;
%replace DTR$M_CONTEXT_SEARCH by '00000000000100000000000000000000'b;
%replace DTR$M_HYPHEN_DISABLED by '00000000000010000000000000000000'b;
%replace DTR$M_MORE_COMMANDS by '00000000000001000000000000000000'b;
%replace DTR$M_ABORT by '00000000000000100000000000000000'b;
%replace DTR$M_LOCK_WAIT by '00000000000000010000000000000000'b;

/* DTR$DTR call options */
%replace DTR$M_OPT_CMD by '10000000000000000000000000000000'b;
%replace DTR$M_OPT_PRMPT by '01000000000000000000000000000000'b;
%replace DTR$M_OPT_LINE by '00100000000000000000000000000000'b;
%replace DTR$M_OPT_MSG by '00010000000000000000000000000000'b;
%replace DTR$M_OPT_PGET by '00001000000000000000000000000000'b;
%replace DTR$M_OPT_PPUT by '00000100000000000000000000000000'b;
%replace DTR$M_OPT_CONT by '00000010000000000000000000000000'b;
%replace DTR$M_OPT_UDK by '00000001000000000000000000000000'b;
%replace DTR$M_OPT_DTR_UDK by '00000000100000000000000000000000'b;
%replace DTR$M_OPT_END_UDK by '00000000010000000000000000000000'b;
%replace DTR$M_OPT_UNWIND by '00000000001000000000000000000000'b;
%replace DTR$M_OPT_CONTROL_C by '00000000000100000000000000000000'b;
%replace DTR$M_OPT_STARTUP by '00000000000010000000000000000000'b;
%replace DTR$M_OPT_FOREIGN by '00000000000001000000000000000000'b;
%replace DTR$M_OPT_BANNER by '00000000000000100000000000000000'b;
%replace DTR$M_OPT_REMOVE_CTLC by '00000000000000010000000000000000'b;
%replace DTR$M_OPT_KEYDEFS by '00000000000000001000000000000000'b;


/* User-defined keyword types */
%replace DTR$K_UDK_SET by 1;
%replace DTR$K_UDK_SET_NO by 2;
%replace DTR$K_UDK_SHOW by 3;
%replace DTR$K_UDK_STATEMENT by 4;
%replace DTR$K_UDK_COMMAND by 5;

/* User-defined keyword token types */
%replace DTR$K_TOK_TOKEN by 1;
%replace DTR$K_TOK_PICTURE by 2;
%replace DTR$K_TOK_FILENAME by 3;
%replace DTR$K_TOK_COMMAND by 4;
%replace DTR$K_TOK_TEST_TOKEN by 5;
%replace DTR$K_TOK_LIST_ELEMENT by 6;
%replace DTR$K_TOK_TEST_EOL by 7;

/* DTR Status messages */
%replace DTR$_SUCCESS by 9274723;

/* DTR$INFO type constants */
%replace DTR$K_INF_TYPE_DOMAIN by 1;
%replace DTR$K_INF_TYPE_COLLECTION by 2;
%replace DTR$K_INF_TYPE_KEYWORD by 3;
%replace DTR$K_INF_TYPE_DIC_NAME by 4;
%replace DTR$K_INF_TYPE_GLV by 5;
%replace DTR$K_INF_TYPE_PLOT by 6;
```

(continued on next page)

```
/* DTR$LOOKUP info constants */
%replace DTR$K_INF_COL_CURSOR by 1;
%replace DTR$K_INF_COL_SIZE by 2;
%replace DTR$K_INF_COL_FLD by 3;
%replace DTR$K_INF_COL_DROPPED by 4;
%replace DTR$K_INF_COL_ERASED by 5;
%replace DTR$K_INF_COL_INVISIBLE by 6;
%replace DTR$K_INF_COL_NAME by 7;
%replace DTR$K_INF_COL_NEXT_COL by 8;
%replace DTR$K_INF_DOM_FLD by 9;
%replace DTR$K_INF_DOM_FORM by 10;
%replace DTR$K_INF_DOM_SHARE by 11;
%replace DTR$K_INF_DOM_ACCESS by 12;
%replace DTR$K_INF_DOM_NAME by 13;
%replace DTR$K_INF_DOM_NEXT_DOM by 14;
%replace DTR$K_INF_DOM_SSC by 15;
%replace DTR$K_INF_FLD_NAME by 16;
%replace DTR$K_INF_FLD_QNAME by 17;
%replace DTR$K_INF_FLD_PICTURE by 18;
%replace DTR$K_INF_FLD_EDIT by 19;
%replace DTR$K_INF_FLD_DTYPE by 20;
%replace DTR$K_INF_FLD_OFFSET by 21;
%replace DTR$K_INF_FLD_LENGTH by 22;
%replace DTR$K_INF_FLD_SCALE by 23;
%replace DTR$K_INF_FLD_CHILD by 24;
%replace DTR$K_INF_FLD_CNT by 25;
%replace DTR$K_INF_FLD_LIST by 26;
%replace DTR$K_INF_FLD_REDEFINES by 27;
%replace DTR$K_INF_FLD_VIRTUAL by 28;
%replace DTR$K_INF_FLD_FILLER by 29;
%replace DTR$K_INF_FLD_MISSING by 30;
%replace DTR$K_INF_FLD_QHDR by 31;
%replace DTR$K_INF_FRM_NAME by 32;
%replace DTR$K_INF_FRM_LIBRARY by 33;
%replace DTR$K_INF_GLV_FIRST_DOM by 34;
%replace DTR$K_INF_GLV_FIRST_COL by 35;
%replace DTR$K_INF_GLV_FIRST_SSC by 36;
%replace DTR$K_INF_GLV_STA_OBJ by 37;
%replace DTR$K_INF_GLV_STA_CNT by 38;
%replace DTR$K_INF_GLV_STA_LINE by 39;
%replace DTR$K_INF_HDR_CNT by 40;
%replace DTR$K_INF_HDR_STRING by 41;
%replace DTR$K_INF_PLO_CNT by 42;
%replace DTR$K_INF_PLO_PAI by 43;
%replace DTR$K_INF_PAI_PROMPT by 44;
%replace DTR$K_INF_PAI_DTYPE by 45;
%replace DTR$K_INF_SSC_NAME by 46;
%replace DTR$K_INF_SSC_SET by 47;
%replace DTR$K_INF_SSC_NEXT by 48;
%replace DTR$K_INF_SET_NAME by 49;
%replace DTR$K_INF_SET_NEXT by 50;
%replace DTR$K_INF_SET_SDP by 51;
%replace DTR$K_INF_SET_SINGULAR by 52;
%replace DTR$K_INF_SDP_NEXT by 53;
%replace DTR$K_INF_SDP_DOMAIN by 54;
%replace DTR$K_INF_SDP_TENANCY by 55;
%replace DTR$K_INF_SDP_INSERT by 56;
%replace DTR$K_INF_SDP_RETAIN by 57;
%replace DTR$K_INF_DOM_REC_LEN by 58;
```

```
/* DTR$LOOKUP answer constants */
%replace DTR$K_INF_DOM_ACCESS_READ by 1;
%replace DTR$K_INF_DOM_ACCESS_WRITE by 2;
%replace DTR$K_INF_DOM_ACCESS_MODIFY by 3;
%replace DTR$K_INF_DOM_ACCESS_EXTEND by 4;

%replace DTR$K_INF_DOM_SHARE_EXCLUSIVE by 1;
%replace DTR$K_INF_DOM_SHARE_SHARED by 2;
%replace DTR$K_INF_DOM_SHARE_PROTECT by 3;

/* Variable declarations section */
DCL 1 DTR_ACCESS_BLOCK based, /*Datatreive Access Block*/
        2 DAB$B_BID fixed binary(7),
        2 DAB$B_BLN fixed binary(7),
        2 DAB$L_CONDITION fixed binary(31),
        2 DAB$A_MSG_BUF pointer,
        2 DAB$W_MSG_BUF_LEN fixed binary(15),
        2 DAB$W_MSG_LEN fixed binary(15),
        2 DAB$A_AUX_BUF pointer,
        2 DAB$W_AUX_BUF_LEN fixed binary(15),
        2 DAB$W_AUX_LEN fixed binary(15),
        2 DAB$W_IDI fixed binary(15),
        2 DAB$W_STATE fixed binary(15),
        2 DAB$L_FLAGS bit(32) aligned,
        2 DAB$L_OPTIONS bit(32) aligned,
        2 DAB$W_REC_LENGTH fixed binary(15),
        2 DAB$W_VERSION fixed binary(15),
        2 DAB$W_LEVEL fixed binary(15),
        2 DAB$B_VER_LETTER character,
        2 DAB$W_BASE_LEVEL fixed binary(15),
        2 DAB$W_UDK_INDEX fixed binary(15),
        2 DAB$W_COLUMNS_PAGE fixed binary(15),
        2 DAB$W_TT_CHANNEL fixed binary(15),
        2 DAB$W_CTLC_CHANNEL fixed binary(15),
        2 DAB$L_KEYTABLE_ID bit(32) aligned,
        2 DAB$L_COMMAND_KEYBOARD bit(32) aligned,
        2 DAB$L_PROMPT_KEYBOARD bit(32) aligned,
        2 DAB$REST_OF_DAB character(37);

DCL DTR$COMMAND entry(
        1 like DTR_ACCESS_BLOCK,                /* dab */
        any character(*),                       /* command-string */
        any descriptor) /* p1...pn */
        returns(fixed binary(31)) options(variable);

DCL DTR$CONTINUE entry(
        1 like DTR_ACCESS_BLOCK)                /* dab */
        returns(fixed binary(31));

DCL DTR$CREATE_UDK entry(
        1 like DTR_ACCESS_BLOCK,                /* dab */
        any character(*),                       /* keyword-name */
        fixed binary(31),                       /* index */
        fixed binary(31))                       /* context */
        returns(fixed binary(31));
```

(continued on next page)

```
DCL DTR$DTR entry(
        1 like DTR_ACCESS_BLOCK,                /* dab */
        bit(32) aligned)                        /* options-code */
        returns(fixed binary(31));

DCL DTR$END_UDK entry(
        1 like DTR_ACCESS_BLOCK)                /* dab */
        returns(fixed binary(31));

DCL DTR$FINISH entry(
        1 like DTR_ACCESS_BLOCK)                /* dab */
        returns(fixed binary(31));

DCL DTR$GET_PORT entry(
        1 like DTR_ACCESS_BLOCK,                /* dab */
        any)                                    /* record-buffer */
        returns(fixed binary(31));

DCL DTR$GET_STRING entry(
        1 like DTR_ACCESS_BLOCK,                /*dab */
        fixed binary(31),                       /* token-type */
        any character(*),                       /* string */
        fixed binary(31),       /* length */
        any character(*))       /* compare-string */
        returns(fixed binary(31)) options(variable);

DCL DTR$INFO entry(
        1 like DTR_ACCESS_BLOCK,                /* dab */
        fixed binary(31),                       /* object-id */
        fixed binary(31),                       /* info-code */
        fixed binary(31),                       /* ret-val */
        any character(*),       /* output-string */
        fixed binary(31) value) /* index */
        returns(fixed binary(31)) options(variable);

DCL DTR$INIT entry(
        1 like DTR_ACCESS_BLOCK,                /* dab */
        fixed binary(15),       /* size */
        character(*),   /* msg-buff */
        character(*),   /* aux-buff */
        bit(32) aligned)        /* options-code */
        returns(fixed binary(31)) options(variable);

DCL DTR$LOOKUP entry(
        1 like DTR_ACCESS_BLOCK,                /* dab */
        fixed binary(31),                       /* object-type */
        fixed binary(31),                       /* object-id */
        any character(*))       /* object-name */
        returns(fixed binary(31)) options(variable);

DCL DTR$PORT_EOF entry(
        1 like DTR_ACCESS_BLOCK)                /* dab */
        returns(fixed binary(31));

DCL DTR$PRINT_DAB entry(
        1 like DTR_ACCESS_BLOCK)                /* dab */
        returns(fixed binary(31));
```

```
DCL DTR$PUT_OUTPUT entry(
        1 like DTR_ACCESS_BLOCK,              /* dab */
        any character(*),                     /* string */
        any character(*))       /* prompt-string */
        returns(fixed binary(31)) options(variable);

DCL DTR$PUT_PORT entry(
        1 like DTR_ACCESS_BLOCK,              /* dab */
        any)                                  /* record-buffer */
        returns(fixed binary(31));

DCL DTR$PUT_VALUE entry(
        1 like DTR_ACCESS_BLOCK,              /* dab */
        any character(*))       /* value */
        returns(fixed binary(31)) options(variable);

DCL DTR$UNWIND entry(
        1 like DTR_ACCESS_BLOCK)              /* dab */
        returns(fixed binary(31));
```

A.6 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, or PL/I, you will have to define the DATATRIEVE Access Block within your program.

To create your own DAB, you must define:

- The access block
- The message buffers
- DATATRIEVE constants

The following sections describe each component of the DATATRIEVE Access Block separately.

A.6.1 Defining the Access Block

The main component of 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-1 describes the data type and purpose of each field.

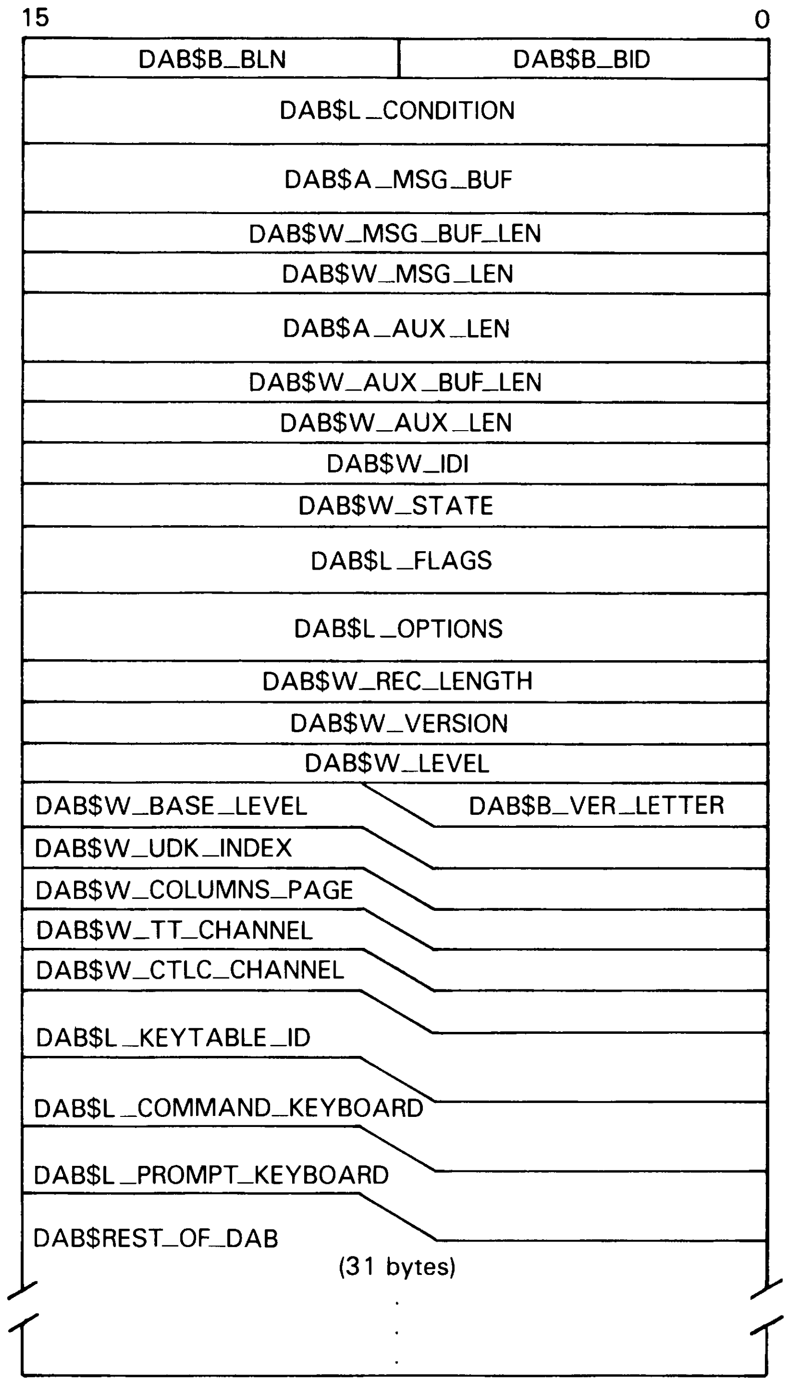

Figure A-1: Structure of the DATATRIEVE Access Block

Table A-1: 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_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

(continued on next page)

Table A-1: Fields of the DATATRIEVE Access Block (Cont.)

DAB Field Name	Data Type	Content of Field
DAB$W_REC_LENGTH	Word integer	Length of the record DATATRIEVE is either ready to pass to or to receive from the calling program (for use with the DTR$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$W_BASE_LEVEL	Word integer	Reserved to DIGITAL
DAB$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, TDMS forms, Guide Mode, and help
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

(continued on next page)

Table A-1: Fields of the DATATRIEVE Access Block (Cont.)

DAB Field Name	Data Type	Content of Field
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 prompts to the terminal (DATATRIEVE uses separate keyboard IDs for command input and prompting)
DAB$REST_OF_DAB	31 bytes (unspecified)	Reserved for future use

A.6.2 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.6.3 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-2.

Table A-2: DATATRIEVE Access Block Constants

DAB Constant	Data Type	Value
Stallpoints		
DTR$K_STL_CMD	Longword integer	1
DTR$K_STL_PRMPT	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
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

(continued on next page)

Table A-2: DATATRIEVE Access Block Constants (Cont.)

DAB Constant	Data Type	Value
Command Options		
DTR$M_OPT_CMD	Longword integer	1
DTR$M_OPT_PRMPT	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_CTLC	Longword integer	32768
DTR$M_OPT_KEYDEFS	Longword integer	64536
User-Defined Keyword Types		
DTR$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

(continued on next page)

Table A-2: DATATRIEVE Access Block Constants (Cont.)

DAB Constant	Data Type	Value
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

DATATRIEVE Message Information B

The documentation of DATATRIEVE messages is now stored on line and may be accessed at the following file specification:

DTR$LIBRARY:DTRMSGS.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 online 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 diplays 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.

Argument Data Types C

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_.

C.1 Atomic Data Types

Symbol	Code	Name/Description
DSC$K_DTYPE_Z	0	Unspecified The calling program has specified no data type. The called procedure should assume the argument is of the correct type.
DSC$K_DTYPE_V	1	Bit An aligned bit string.
DSC$K_DTYPE_BU	2	Byte logical An 8-bit unsigned quantity.

(continued on next page)

Symbol	Code	Name/Description
DSC$K_DTYPE_WU	3	Word logical A 16-bit unsigned quantity.
DSC$K_DTYPE_LU	4	Longword logical A 32-bit unsigned quantity.
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.

(continued on next page)

Symbol	Code	Name/Description
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$K_DTYPE_DC	13	D_floating complex An ordered pair of D_floating quantities, representing a double-precision complex number. The lower addressed quantity is the real part; the higher addressed quantity is the imaginary part.
DSC$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.

C.2 String Data Types

The following string types are ordinarily described by a string descriptor.

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

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)

Symbol	Code	Name/Description
DSC$K_DTYPE_BPV	32	Bound procedure value A two longword entity in which the first longword contains the address of a procedure entry mask and the second longword is the environment value. The environment value is determined in a language-specific manner when the original bound procedure value is generated. When the bound procedure is called, the calling program loads the second longword into R1. When the environment value is not needed, this data type can be passed using the immediate value mechanism. In this case, the argument list entry contains the address of the procedure entry mask and the second longword is omitted.
DSC$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

This index uses the following symbols:

t	Entry occurrence in a table
f	Entry occurrence in a figure
e	Entry occurrence in an example

How to Order Additional Documentation

If you live in:	Call:	or Write:
New Hampshire, Alaska	603-884-6660	Digital Equipment Corp. P.O. Box CS2008 Nashua, NH 03061-2698
Continental USA, Puerto Rico, Hawaii	1-800-258-1710	Same as above.
Canada (Ottawa-Hull)	613-234-7726	Digital Equipment Corp. 940 Belfast Road Ottawa, Ontario K1G 4C2 Attn: P&SG Business Manager or approved distributor
Canada (British Columbia)	1-800-267-6146	Same as above.
Canada (All other)	112-800-267-6146	Same as above.
All other areas	—	Digital Equipment Corp. Peripherals & Supplies Centers P&SG Business Manager c/o DIGITAL's local subsidiary

Note: Place prepaid orders from Puerto Rico with the local DIGITAL subsidiary (phone 809-754-7575).

Place internal orders with the Software Distribution Center, Digital Drive, Westminster, MA 01473-0471.

Reader's Comments

VAX DATATRIEVE
Guide to Programming and Customizing
AA–P863C–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less ______________________________

____

What I like best about this manual is ______________________________

____

What I like least about this manual is ______________________________

____

I found the following errors in this manual:

Page	Description
______	______________________________
______	______________________________
______	______________________________

Additional comments or suggestions to improve this manual:

____

____

____

I am using **Version** ______ of the software this manual describes.

Name/Title ______________________________ Dept. ______________

Company ______________________________ Date ______________

Mailing Address ______________________________

______________________________ Phone ______________

— — Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35
110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

— — Do Not Tear - Fold Here

Cut Along Dotted Line

Reader's Comments

VAX DATATRIEVE
Guide to Programming and Customizing
AA–P863C–TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less ______________________________

____

What I like best about this manual is ______________________________

____

What I like least about this manual is ______________________________

____

I found the following errors in this manual:
Page Description

_____ ______________________________

_____ ______________________________

_____ ______________________________

Additional comments or suggestions to improve this manual:

____

____

____

I am using **Version** _____ of the software this manual describes.

Name/Title ______________________ Dept. ____________

Company ______________________ Date ____________

Mailing Address ______________________________

______________________ Phone ____________

—— Do Not Tear - Fold Here and Tape ——————————

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35
110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

——— Do Not Tear - Fold Here ——————————

Cut Along D[illegible]d Line

digital™

DIGITAL EQUIPMENT CORPORATION

Printed in U.S.A.